

Intensional RDB Manifesto: a Unifying NewSQL Model for Flexible Big Data

Zoran Majkić

ISRST, Tallahassee, FL, USA
majk.1234@yahoo.com
<http://zoranmajkic.webs.com/>

Abstract. In this paper we present a new family of Intensional RDBs (IRDBs) which extends the traditional RDBs with the Big Data and flexible and 'Open schema' features, able to preserve the user-defined relational database schemas and all preexisting user's applications containing the SQL statements for a deployment of such a relational data. The standard RDB data is parsed into an internal vector key/value relation, so that we obtain a column representation of data used in Big Data applications, covering the key/value and column-based Big Data applications as well, into a unifying RDB framework. We define a query rewriting algorithm, based on the GAV Data Integration methods, so that each user-defined SQL query is rewritten into a SQL query over this vector relation, and hence the user-defined standard RDB schema is maintained as an empty global schema for the RDB schema modeling of data and as the SQL interface to stored vector relation. Such an IRDB architecture is adequate for the massive migrations from the existing slow RDBMSs into this new family of fast IRDBMSs by offering a Big Data and new flexible schema features as well.

1 Introduction

The term NoSQL was picked out in 2009 and used for conferences of advocates of non-relational databases. In an article of the Computerworld magazine [1], June 2009, dedicated to the NoSQL meet-up in San Francisco is reported the following: "NoSQLers came to share how they had overthrown the tyranny of slow, expensive relational databases in favor of more efficient and cheaper ways of managing data". In this article, the Computerworld summarizes the following reasons:

- *High Throughput.* The NoSQL databases provide a significantly higher data throughput than traditional RDBMSs.
- *Horizontal Scalability.* In contrast to RDBMSs most NoSQL databases are designed to scale well in the horizontal direction and not rely on highly available hardware.
- *Cost Setting and Complexity of Database Clusters.* NoSQL does not need the complexity and cost of sharding which involves cutting up databases into multiple tables to run on large clusters or grids.
- *"One size fits all" [2] Database Thinking Is Wrong.* The thinks that the realization and the search for alternatives towards traditional RDBMs can be explained by the following two major trends: The continuous growth of data volumes and the growing need to process larger amounts of data in shorter time.

- *Requirement of Cloud Computing.* Are mentioned two major requirements: High until almost ultimate scalability (especially in the horizontal direction) and low administration overhead. Developed cloud Amazon's SimpleDB can store large collections of items which themselves are hashtables containing attributes that consist of key-value pairs.
- *Avoidance of Unneeded Complexity.* The reach feature set and the ACID properties implemented by RDBMSs might be more than necessary for particular applications. There are different scenarios where applications would be willing to compromise reliability for better performances.

Moreover, the NoSQL movements advocate that relational fit well for data that is *rigidly structured* with relations and are designated for central deployments with single, large high-end machines, and not for distribution. Often they emphasize that SQL queries are expressed in a sophisticated language. But usually they do not tell that also the NoSQL databases often need the sophisticated languages (as object-oriented databases, or Graph-based databases) as well. Moreover, they do not say that SQL and RDB are based on sound logical framework (a subset of the First-Order Logic (FOL) language) and hence it is not a procedural language, but a higher level declarative language able to specify "what" we need instead of "how" to obtain what we need. Thus, from the point of view of the development of computer science, instead to go in the direction of the logically higher levels of knowledge representation and query languages, more appropriated to the human understanding, they propose the old technics as key-value representations or the simpler forms of object-oriented representations and their relative procedural query languages.

They jumped into the past instead to jump in the future of the social and scientific human development. It happened because the current RDBMSs were obsolete and not ready to accept the new social-network Web applications in the last 10 years, so that the isolated groups of developers of these ad-hoc systems (e.g., Google, Amazon, LinkedIn, Facebook, etc..) could use only the ready old-known technics and development instruments in order to satisfy the highly urgent business market requirements. From the academic research side, instead, most of the work has been done in Sematic Web "industrial-funded" programs (e.g., the European IST projects) by considering the new knowledge and reasoning logic systems, whose impact to the existing RDB applications framework would be very hard, with difficult migration and implementation in these new semantics systems (it would need one or more decade of time). Instead, we needed a more fundamental theoretical research for the significative technological advances and evolutions of the existing RDB engine. Thus, the core RDB technology was in some way "abandoned" from both major development initiatives in the last 20 years. Nobody probably wanted to consider the most natural evolution of the RDBMSs and its FOL and SQL query framework, and the database industry tried only to cover "with pieces" and "adding" the new emergent customer's necessities, without a strong investment and the necessary efforts for the complete revision of their old System R based RDB engines of the 1970s. The world's economical crisis form 2007 did not help for such an effort.

However, from the technical point of view, it is clear that if we would come back to make the application programs in Assembler, we probably will obtain better computa-

tional performances for some algorithms than with more powerful programming languages, but it is justifiable when we write the system infrastructures and parsers, and not when we have to develop the legacy software for user's requirements. Analogously, we may provide the BD infrastructure and physical level in a form of simpler structures, adequate to support the distributive and massive BigData query computations, by preserving the logically higher level interface to customer's applications. That is, it is possible to preserve the RDB interface to data, with SQL query languages for the programmers of the software applications, with the "physical" parsing of data in more simple structures, able to deal with Big Data scalability in a high distributive computation framework.

The first step to maintain the logical declarative (non-procedural) SQL query language level, is done by the group (M.I.T. and Microsoft) and in widely adopted paper "The End of an Architectural Era" (cf. [3] Michael Stonebraker et al.) where the authors come to the conclusion "that the current RDBMS code lines, while attempting to be a "one size fits all" solution, in fact excel at nothing". At first, Stonebraker et al. argue that RDBMSs have been architected more than 25 years ago when the hardware characteristics, user requirements and database markets were very different from those today. The resulting revision of traditional RDBMSs is provided by developing H-store (M.I.T., Brown and Yale University), a next generation OLTP systems that operates on distributed clusters of shared-nothing machines where the data resides entirely in main memory, so that it was shown to significantly outperform (83 times) a traditional, disc-based DBMS. A more full-featured version of the system [4] that is able to execute across multiple machines within a local area cluster has been presented in August 2008. The data storage in H-store is managed by a single-thread execution engine that resides underneath the transaction manager. Each individual site executes an autonomous instance of the storage engine with a fixed amount of memory allocated from its host machine. Multi-site nodes do not share any data structures with collocated sites, and hence there is no need to use concurrent data structures (every read-only table is replicated on all nodes and other tables are divided horizontally into disjoint partitions with a k-safety factor two). Thus, H-store (at <http://hstore.cs.brown.edu/documentation/architecture-overview/>) was designed as a parallel, row-storage relational DBMS that runs on a cluster of shared-nothing, main memory executor nodes. The commercial version of H-store's design is VoltDB.

More recently, during 2010 and 2011, Stonebraker has been a critic of the NoSQL movement [5,6]: "Here, we argue that using MR systems to perform tasks that are best suited for DBMSs yields less than satisfactory results [7], concluding that MR is more like an extract-transform-load (ETL) system than a DBMS, as it quickly loads and processes large amounts of data in an ad hoc manner. As such, it complements DBMS technology rather than competes with it." After a number of arguments about MR (MapReduction) w.r.t. SQL (with GROUP BY operation), the authors conclude that parallel DBMSs provide the same computing model as MR (popularized by Google and Hadoop to process key/value data pairs), with the added benefit of using a declarative SQL language. Thus, parallel DBMSs offer great scalability over the range of nodes that customers desire, where all parallel DBMSs operate (pipelining) by creating a query plan that is distributed to the appropriate nodes at execution time. When one operator in

this plan send data to next (running on the same or a different node), the data are pushed by the first to the second operator (this concept is analog to the process described in my book [8], February 2014, in Section 5.2.1 dedicated to normalization of SQL terms (completeness of the Action-relational-algebra category **RA**), so that (differently from MR), the intermediate data is never written to disk. The formal theoretical framework (the database category **DB**) of the parallel DBMSs and the semantics of database mappings between them is provided in Big Data integration theory as well [8].

It is interesting that in [6], the authors conclude that parallel DBMSs excel at efficient querying of large data sets while MR key/value style systems excel at complex analytics and ETL tasks, and propose: "The result is a much more efficient overall system than if one tries to do the entire application in either system. That is, smart software is always a good idea."

The aim of this paper is to go one step in advance in developing this NewSQL approach, and to extend the "classic" RDB systems with both features: to offer, on user's side, the standard RDB database schema for SQL querying and, on computational side, the "vectorial" relational database able to efficiently support the low-level key/value data structures together, in the same logical SQL framework. Moreover, this parsing of the standard RDBs into a "vectorial" database efficiently resolves also the problems of NoSQL applications with sparse-matrix and "Open schema" data models.

The plan of this paper is the following: In Section 2 we present the method of parsing of any RDB into a vector relation of the key/value structure, compatible with most Big Data structures, and Open schema solutions, but with preserving the RDB user-defined schema for the software applications. We show that such a parsing changes the standard semantics of the RDBs based on the FOL by introducing the intensional concepts for user-defined relational tables. Consequently, in Section 3 we introduce a *conservative intensional extension* of the FOL adequate to express the semantics for the IRDBs and the SQL. In Section 4 we define a new semantics for the IRDBs and their canonical models based on the Data Integration systems, where the user-defined RDB is a global schema and the source schema is composed by the unique vector relations which contains the parsed data of the whole used-defined RDB. As in GAV (Global-As-View) Data Integration systems, we dematerialize the global schema (i.e., user-defined RDB) and, in Section 5, we define a *query-rewriting algorithm* to translate the original query written for the user-defined RDB schema into the source database composed by the vector relation containing the parsed data.

2 Vector databases with intensional FOL Semantics

In what follows, we denote by B^A the set of all functions from A to B , and by A^n a n -folded cartesian product $A \times \dots \times A$ for $n \geq 1$, we denote by $\neg, \wedge, \vee, \Rightarrow$ and \Leftrightarrow the logical operators negation, conjunction, disjunction, implication and equivalence, respectively. For any two logical formulae ϕ and ψ we define the XOR logical operator $\underline{\vee}$ by $\phi \underline{\vee} \psi$ logically equivalent to $(\phi \vee \psi) \wedge \neg(\phi \wedge \psi)$. Then we will use the following RDB definitions, based on the standard First-Order Logic (FOL) semantics:

- A *database schema* is a pair $\mathcal{A} = (S_A, \Sigma_A)$ where S_A is a countable set of relational symbols (predicates in FOL) $r \in \mathbb{R}$ with finite arity $n = ar(r) \geq 1$

($ar : \mathbb{R} \rightarrow \mathcal{N}$), disjoint from a countable infinite set **att** of attributes (a domain of $a \in \mathbf{att}$ is a nonempty finite subset $dom(a)$ of a countable set of individual symbols **dom**). For any $r \in \mathbb{R}$, the sort of r , denoted by tuple $\mathbf{a} = atr(r) = \langle atr_r(1), \dots, atr_r(n) \rangle$ where all $a_i = atr_r(m) \in \mathbf{att}$, $1 \leq m \leq n$, must be distinct: if we use two equal domains for different attributes then we denote them by $a_i(1), \dots, a_i(k)$ (a_i equals to $a_i(0)$). Each index ("column") i , $1 \leq i \leq ar(r)$, has a distinct column name $nr_r(i) \in SN$ where SN is the set of names with $nr(r) = \langle nr_r(1), \dots, nr_r(n) \rangle$. A relation symbol $r \in \mathbb{R}$ represents the *relational name* and can be used as an atom $r(\mathbf{x})$ of FOL with variables in \mathbf{x} assigned to its columns, so that Σ_A denotes a set of sentences (FOL formulae without free variables) called *integrity constraints* of the sorted FOL with sorts in **att**.

- An *instance-database* of a nonempty schema \mathcal{A} is given by $A = (\mathcal{A}, I_T) = \{R = \|r\| = I_T(r) \mid r \in S_A\}$ where I_T is a Tarski's FOL interpretation which satisfies *all* integrity constraints in Σ_A and maps a relational symbol $r \in S_A$ into an n -ary relation $R = \|r\| \in A$. Thus, an instance-database A is a set of n -ary relations, managed by relational database systems.

Let A and $A' = (\mathcal{A}, I'_T)$ be two instances of \mathcal{A} , then a function $h : A \rightarrow A'$ is a *homomorphism* from A into A' if for every k -ary relational symbol $r \in S_A$ and every tuple $\langle v_1, \dots, v_k \rangle$ of this k -ary relation in A , $\langle h(v_1), \dots, h(v_k) \rangle$ is a tuple of the same symbol r in A' . If A is an instance-database and ϕ is a sentence then we write $A \models \phi$ to mean that A satisfies ϕ . If Σ is a set of sentences then we write $A \models \Sigma$ to mean that $A \models \phi$ for every sentence $\phi \in \Sigma$. Thus the set of all instances of \mathcal{A} is defined by $Inst(\mathcal{A}) = \{A \mid A \models \Sigma_A\}$.

- We consider a rule-based *conjunctive query* over a database schema \mathcal{A} as an expression $q(\mathbf{x}) \leftarrow r_1(\mathbf{u}_1), \dots, r_n(\mathbf{u}_n)$, with finite $n \geq 0$, r_i are the relational symbols (at least one) in \mathcal{A} or the built-in predicates (e.g. $\leq, =$, etc.), q is a relational symbol not in \mathcal{A} and \mathbf{u}_i are free tuples (i.e., one may use either variables or constants). Recall that if $\mathbf{v} = (v_1, \dots, v_m)$ then $r(\mathbf{v})$ is a shorthand for $r(v_1, \dots, v_m)$. Finally, each variable occurring in \mathbf{x} is a *distinguished* variable that must also occur at least once in $\mathbf{u}_1, \dots, \mathbf{u}_n$. Rule-based conjunctive queries (called rules) are composed of a subexpression $r_1(\mathbf{u}_1), \dots, r_n(\mathbf{u}_n)$ that is the *body*, and the *head* of this rule $q(\mathbf{x})$. The *Yes/No* conjunctive queries are the rules with an empty head. If we can find values for the variables of the rule, such that the body is logically satisfied, then we can deduce the head-fact. This concept is captured by a notion of "valuation".

The deduced head-facts of a conjunctive query $q(\mathbf{x})$ defined over an instance A (for a given Tarski's interpretation I_T of schema \mathcal{A}) are equal to $\|q(x_1, \dots, x_k)\|_A = \{ \langle v_1, \dots, v_k \rangle \in \mathbf{dom}^k \mid A \models \exists \mathbf{y} (r_1(\mathbf{u}_1) \wedge \dots \wedge r_n(\mathbf{u}_n)) [x_i/v_i]_{1 \leq i \leq k} \} = I_T^*(\exists \mathbf{y} (r_1(\mathbf{u}_1) \wedge \dots \wedge r_n(\mathbf{u}_n)))$, where the \mathbf{y} is a set of variables which are not in the head of query, and I_T^* is the unique extension of I_T to all formulae. We recall that the conjunctive queries are monotonic and satisfiable, and that a (Boolean) query is a class of instances that is closed under isomorphism [9]. Each conjunctive query corresponds to a "select-project-join" term $t(\mathbf{x})$ of SPRJU algebra obtained from the formula $\exists \mathbf{y} (r_1(\mathbf{u}_1) \wedge \dots \wedge r_n(\mathbf{u}_n))$, as explained in Section 5.

- We consider a finitary *view* as a union of a finite set S of conjunctive queries with the same head $q(\mathbf{x})$ over a schema \mathcal{A} , and from the equivalent algebraic point of view, it is a "select-project-join-rename + union" (SPJRU) finite-length term $t(\mathbf{x})$

which corresponds to union of the terms of conjunctive queries in S . In what follows we will use the same notation for a FOL formula $q(\mathbf{x})$ and its equivalent algebraic SPJRU expression $t(\mathbf{x})$. A materialized view of an instance-database A is an n -ary relation $R = \bigcup_{q(\mathbf{x}) \in S} \|q(\mathbf{x})\|_A$. Notice that a finitary view can also have an infinite number of tuples. We denote the set of all finitary materialized views that can be obtained from an instance A by TA .

The principal idea is to use an analogy with a GAV Data Integration [10,8] by using the database schema $\mathcal{A} = (S_A, \Sigma_A)$ as a global relational schema, used as a user/application-program interface for the query definitions in SQL, and to represent the source database of this Data Integration system by parsing of the RDB instance A of the schema \mathcal{A} into a single vector relation \vec{A} . Thus, the original SQL query $q(\mathbf{x})$ has to be equivalently rewritten over (materialized) source vector database \vec{A} .

The idea of a vector relation \vec{A} for a given relational database instance A comes from the investigation of the topological properties of the RDB systems, presented in Chapter 8 of the book [8]. In order to analyze the algebraic lattice of all RDB database instances each instance database A , composed by a set of finitary relations $R_i \in A$, $i = 1, \dots, n$, in Lemma 21 is defined the transformation of the instance database A into a vector relation \vec{A} , with $\vec{A} = \bigcup_{R \in A} \vec{R}$ where for each relation R $ar(R) \geq 1$ is the arity (the number of its columns) of this relational table and π_i is its i -th column projection, and hence $\vec{R} = \bigcup_{1 \leq i \leq ar(R)} \pi_i(R)$. Such vectorial representation of a given database A in [8] is enough to define the lattice of RDB lattices, because we do not needed the converse process (to define a database A from its vectorial representation).

However, by considering that a database A is seen by the users and their software applications (with the embedded SQL statements), while \vec{A} is its single-table internal representation, over which is executed a rewritten user's query, the extracted information has to be converted in the RDB form w.r.t. the relational schema of the original user's model. Consequently, we need a reacher version of the vector database, such that we can obtain an equivalent inverse transformation of it into the standard user defined RDB schema.

In fact, each i -th column value d_i in a tuple $\mathbf{d} = \langle d_1, \dots, d_i, \dots, d_{ar(r)} \rangle$ of a relation $R_k = \|r_k\|$, $r_k \in S_A$, of the instance database A is determined by the free dimensional coordinates: relational name $nr(r)$, the attribute name $nr_r(i)$ of the i -th column, and the tuple index $Hash(\mathbf{d})$ obtained by hashing the string of the tuple \mathbf{d} . Thus, the relational schema of the vector relation is composed by the four attributes, relational name, tuple-index, attribute name, and value, i.e., r -name, t -index, a -name and $value$, respectively, so that if we assume r_V (the name of the database \mathcal{A}) for the name of this vector relation \vec{A} then this relation can be expressed by the quadruple $r_V(r\text{-name}, t\text{-index}, a\text{-name}, value)$,

and the parsing of any RDB instance A of a schema \mathcal{A} can be defined as:

Definition 1. PARSING RDB INSTANCES:

Given a database instance $A = \{R_1, \dots, R_n\}$, $n \geq 1$, of a RDB schema $\mathcal{A} = (S_A, \Sigma_A)$ with $S_A = \{r_1, \dots, r_n\}$ such that $R_k = \|r_k\|$, $k = 1, \dots, n$, then the extension $\vec{A} = \|r_V\|$ of the vector relational symbol (name) r_V with the schema $r_V(r\text{-name}, t\text{-index}, a\text{-name}, value)$, and NOT NULL constraints for all its four attributes, and with the

primary key composed by the first three attributes, is defined by:

we define the operation PARSE for a tuple $\mathbf{d} = \langle d_1, \dots, d_{ar(r_k)} \rangle$ of the relation $r_k \in S_A$ by the mapping

$(r_k, \mathbf{d}) \mapsto \{ \langle r_k, Hash(\mathbf{d}), nr_{r_k}(i), d_i \rangle \mid d_i \text{ NOT NULL}, 1 \leq i \leq ar(r_k) \}$, so that

(1) $\vec{A} = \bigcup_{r_k \in S_A, \mathbf{d} \in ||r_k||} \text{PARSE}(r_k, \mathbf{d})$.

Based on the vector database representation $||r_V||$ we define a GAV Data Integration system $\mathcal{I} = \langle \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ with the global schema $\mathcal{A} = (S_A, \Sigma_A)$, the source schema $\mathcal{S} = (\{r_V\}, \emptyset)$, and the set of mappings \mathcal{M} expressed by the tgds (tuple generating dependencies)

(2) $\forall y, x_1, \dots, x_{ar(r_k)} ((r_V(r_k, y, nr_{r_k}(1), x_1) \sqcup x_1 \text{ NULL}) \wedge \dots \wedge (r_V(r_k, y, nr_{r_k}(ar(r_k)), x_{ar(r_k)}) \sqcup x_{ar(r_k)} \text{ NULL})) \Rightarrow r_k(x_1, \dots, x_{ar(r_k)}))$,

for each $r_k \in S_A$.

The operation PARSE corresponds to the parsing of the tuple \mathbf{v} of the relation $r_k \in S_A$ of the user-defined database schema \mathcal{A} into a number of tuples of the vector relation r_V . In fact, we can use this operation for virtual inserting/deleting of the tuples in the user defined schema \mathcal{A} , and store them only in the vector relation r_V . This operation avoids to materialize the user-defined (global) schema, but only the source database \mathcal{S} , so that each user-defined SQL query has to be equivalently rewritten over the source database (i.e., the big table $\vec{A} = ||r_V||$) as in standard FOL Data Integration systems.

Notice that this parsing defines a kind of GAV Data Integration systems, where the source database \mathcal{S} is composed by the unique vector relation $||r_V|| = \vec{A}$ (Big Data) which does not contain NULL values, so that we do not unnecessarily save the NULL values of the user-defined relational tables $r_k \in S_A$ in the main memories of the parallel RDBMS used to horizontal partitioning of the unique big-table \vec{A} . Moreover, any adding of the new columns to the user-defined schema \mathcal{A} does not change the table \vec{A} , while the deleting of a i -th column of a relation r will delete all tuples $r_V(x, y, z, v)$ where $x = nr(r)$ and $z = nr_r(i)$ in the main memory of the parallel RDBMS. Thus, we obtain very *schema-flexible* RDB model for Big Data.

Other obtained NoSQL systems properties are:

- *Compatible with key/value systems.* Note that the vector big-table \vec{A} is in the 6th normal form, that is with the primary key corresponding to the first three attributes (the free dimensional coordinates) and the unique value attribute. Thus we obtained the key/value style used for NoSQL Big Data systems. That is, the RDB parsing with resulting Data Integration system subsumes all Big Data key/value systems.
- *Compatible with "Open schema" systems.* Entity-attribute-value model (EAV) is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest. In mathematics, this model is known as a sparse matrix. EAV is also known as object-attribute-value model, vertical database model and open schema. We can use the special relational symbol with name "OpenSchema" in the user database schema \mathcal{A} so that its tuples in \vec{A} will corresponds to atoms $r_V(\text{OpenSchema}, \text{object}, \text{attribute}, \text{value})$. In this case the software developed for the applications which use the Open schema data

will directly access to the vector relation \vec{A} and DBMS will restrict all operations only to tuples where the first attribute has the value equal to OpenSchema (during an inserting of a new tuple $\langle object, attribute, value \rangle$ the DBMS inserts also the value OpenSchema in the first column of \vec{A}).

But this simple and unifying framework needs more investigation for the SQL and underlying logical framework. In fact, we can easily see that the mapping tgds used from the Big Data vector table \vec{A} (the source schema in Data Integration) into user-defined RDB schema \mathcal{A} (the global schema of this Data Integration system with integrity constraints) is not simple FOL formula. Because the same element r_k is used as a predicate symbol (on the right-side of the tgd's implication) and as a value (on the left side of the implication as the first value in the predicate r_V). It means that the elements of the domain of this logic are the elements of other classes and are the classes for themselves as well. Such semantics is not possible in the standard FOL, but only in the *intensional* FOL, and hence the Data Integration \mathcal{I} is not a classic FOL Data Integration as in [10] but an Intensional Data Integration system. In the next sections we will investigate what is the proper logic framework for this class of RDBs, denominated as IRDBs (Intensional RDBs), and to show that the standard SQL is complete in this new logical framework.

3 Conservative intensional extension of the FOL for IRDBs

The first conception of intensional entities (or concepts) is built into the *possible-worlds* treatment of Properties, Relations and Propositions (PRP)s. This conception is commonly attributed to Leibniz, and underlies Alonzo Church's alternative formulation of Frege's theory of senses ("A formulation of the Logic of Sense and Denotation" in Henle, Kallen, and Langer, 3-24, and "Outline of a Revised Formulation of the Logic of Sense and Denotation" in two parts, *Nous*, VII (1973), 24-33, and VIII, (1974), 135-156). This conception of PRPs is ideally suited for treating the *modalities* (necessity, possibility, etc..) and to Montague's definition of intension of a given virtual predicate $\phi(x_1, \dots, x_k)$ (a FOL open-sentence with the tuple of free variables (x_1, \dots, x_k)), as a mapping from possible worlds into extensions of this virtual predicate. Among the possible worlds we distinguish the *actual* possible world. For example, if we consider a set of predicates, of a given Database, and their extensions in different time-instances, then the actual possible world is identified by the current instance of the time.

The second conception of intensional entities is to be found in Russell's doctrine of logical atomism. In this doctrine it is required that all complete definitions of intensional entities be finite as well as unique and non-circular: it offers an *algebraic* way for definition of complex intensional entities from simple (atomic) entities (i.e., algebra of concepts), conception also evident in Leibniz's remarks. In a predicate logics, predicates and open-sentences (with free variables) expresses classes (properties and relations), and sentences express propositions. Note that classes (intensional entities) are *reified*, i.e., they belong to the same domain as individual objects (particulars). This endows the intensional logics with a great deal of uniformity, making it possible to manipulate classes and individual objects in the same language. In particular, when viewed as an individual object, a class can be a member of another class.

The distinction between intensions and extensions is important (as in lexicography [11]), considering that extensions can be notoriously difficult to handle in an efficient manner. The extensional equality theory of predicates and functions under higher-order semantics (for example, for two predicates with the same set of attributes $p = q$ is true iff these symbols are interpreted by the same relation), that is, the strong equational theory of intensions, is not decidable, in general. For example, the second-order predicate calculus and Church's simple theory of types, both under the standard semantics, are not even semi-decidable. Thus, separating intensions from extensions makes it possible to have an equational theory over predicate and function names (intensions) that is separate from the extensional equality of relations and functions.

Relevant recent work about the intension, and its relationship with FOL, has been presented in [12] in the consideration of rigid and *non-rigid* objects, w.r.t. the possible worlds, where the rigid objects, like "George Washington", are the same things from possible world to possible world. Non-rigid objects, like "the Secretary-General of United Nations", are varying from circumstance to circumstance and can be modeled semantically by functions from possible worlds to domain of rigid objects, like intensional entities. However, Fitting substantially and ad-hock changes the syntax and semantics of FOL, and introduces the Higher-order Modal logics, differently from our approach. More about other relevant recent works are presented in [13,14] where a new conservative intensional extension of the Tarski's semantics of the FOL is defined.

Intensional entities are such concepts as propositions and properties. The term 'intensional' means that they violate the principle of extensionality; the principle that extensional equivalence implies identity. All (or most) of these intensional entities have been classified at one time or another as kinds of Universals [15].

We consider a non empty domain $\mathcal{D} = D_{-1} \cup D_I$, where a subdomain D_{-1} is made of particulars (extensional entities), and the rest $D_I = D_0 \cup D_1 \dots \cup D_n \dots$ is made of universals (D_0 for propositions (the 0-ary concepts), and $D_n, n \geq 1$, for n-ary concepts). The fundamental entities are *intensional abstracts* or so called, 'that'-clauses. We assume that they are singular terms; Intensional expressions like 'believe', 'mean', 'assert', 'know', are standard two-place predicates that take 'that'-clauses as arguments. Expressions like 'is necessary', 'is true', and 'is possible' are one-place predicates that take 'that'-clauses as arguments. For example, in the intensional sentence "it is necessary that ϕ ", where ϕ is a proposition, the 'that ϕ ' is denoted by the $\langle\phi\rangle$, where $\langle\rangle$ is the intensional abstraction operator which transforms a logic formula into a *term*. Or, for example, "x believes that ϕ " is given by formula $p_i(x, \langle\phi\rangle)$ (p_i is binary 'believe' predicate). We introduce an intensional FOL [14], with slightly different intensional abstraction than that originally presented in [16], as follows:

Definition 2. *The syntax of the First-order Logic (FOL) language \mathcal{L} with intensional abstraction $\langle\rangle$ is as follows:*

Logical operators (\wedge, \neg, \exists); Predicate letters $r_i, p_i \in \mathbb{R}$ with a given arity $k_i = ar(r_i) \geq 1, i = 1, 2, \dots$ (the functional letters are considered as particular case of the predicate letters); a set PR of propositional letters (nullary predicates) with a truth $r_0 \in PR \cap \mathbb{R}$; Language constants $\bar{0}, \bar{1}, \dots, \bar{c}, \bar{d}, \dots$; Variables x, y, z, \dots in \mathcal{V} ; Abstraction $\langle\rangle$, and punctuation symbols (comma, parenthesis). With the following simultaneous inductive definition of term and formula:

1. All variables and constants are terms. All propositional letters are formulae.
2. If t_1, \dots, t_k are terms then $r_i(t_1, \dots, t_k)$ is a formula for a k -ary predicate letter $r_i \in \mathbb{R}$.
3. If ϕ and ψ are formulae, then $(\phi \wedge \psi)$, $\neg\phi$, and $(\exists x)\phi$ are formulae.
4. If $\phi(\mathbf{x})$ is a formula (virtual predicate) with a list of free variables in $\mathbf{x} = (x_1, \dots, x_n)$ (with ordering from-left-to-right of their appearance in ϕ), and α is its sublist of distinct variables, then $\langle\phi\rangle_\alpha^\beta$ is a term, where β is the remaining list of free variables preserving ordering in \mathbf{x} as well. The externally quantifiable variables are the free variables not in α . When $n = 0$, $\langle\phi\rangle$ is a term which denotes a proposition, for $n \geq 1$ it denotes a n -ary concept.

An occurrence of a variable x_i in a formula (or a term) is bound (free) iff it lies (does not lie) within a formula of the form $(\exists x_i)\phi$ (or a term of the form $\langle\phi\rangle_\alpha^\beta$ with $x_i \in \alpha$). A variable is free (bound) in a formula (or term) iff it has (does not have) a free occurrence in that formula (or term). A sentence is a formula having no free variables.

An interpretation (Tarski) I_T consists of a nonempty domain $\mathcal{D} = D_{-1} \cup D_I$ and a mapping that assigns to any predicate letter $r_i \in \mathbb{R}$ with $k = ar(r_i) \geq 1$, a relation $\|r_i\| = I_T(r_i) \subseteq \mathcal{D}^k$; to each individual constant \bar{c} one given element $I_T(\bar{c}) \in \mathcal{D}$, with $I_T(\bar{0}) = 0$, $I_T(\bar{1}) = 1$ for natural numbers $\mathcal{N} = \{0, 1, 2, \dots\}$, and to any propositional letter $p \in PR$ one truth value $I_T(p) \in \{f, t\}$, where f and t are the empty set $\{\}$ and the singleton set $\{\langle\rangle\}$ (with the empty tuple $\langle\rangle \in D_{-1}$), as those used in the Codd's relational-database algebra [17] respectively, so that for any I_T , $I_T(r_\emptyset) = \{\langle\rangle\}$ (i.e., r_\emptyset is a tautology), while $Truth \in D_0$ denotes the concept (intension) of this tautology. Note that in the intensional semantics a k -ary functional symbol, for $k \geq 1$, in standard (extensional) FOL is considered as a $(k + 1)$ -ary predicate symbols: let f_m be such a $(k + 1)$ -ary predicate symbol which represents a k -ary function denoted by \underline{f}_m with standard Tarski's interpretation $I_T(\underline{f}_m) : \mathcal{D}^k \rightarrow \mathcal{D}$. Then $I_T(f_m)$ is a relation obtained from its graph, i.e., $I_T(f_m) = R = \{(d_1, \dots, d_k, I_T(\underline{f}_m)(d_1, \dots, d_k)) \mid d_i \in \mathcal{D}, 1 \leq i \leq k\}$. The universal quantifier is defined by $\forall = \neg\exists\neg$. Disjunction $\phi \vee \psi$ and implication $\phi \Rightarrow \psi$ are expressed by $\neg(\neg\phi \wedge \neg\psi)$ and $\neg\phi \vee \psi$, respectively. In FOL with the identity \doteq , the formula $(\exists_1 x)\phi(x)$ denotes the formula $(\exists x)\phi(x) \wedge (\forall x)(\forall y)(\phi(x) \wedge \phi(y) \Rightarrow (x \doteq y))$. We denote by R_\doteq the Tarski's interpretation of \doteq . In what follows any open-sentence, a formula ϕ with non empty tuple of free variables (x_1, \dots, x_m) , will be called a m -ary *virtual predicate*, denoted also by $\phi(x_1, \dots, x_m)$. This definition contains the precise method of establishing the *ordering* of variables in this tuple: such an method that will be adopted here is the ordering of appearance, from left to right, of free variables in ϕ . This method of composing the tuple of free variables is the unique and canonical way of definition of the virtual predicate from a given formula.

An *intensional interpretation* of this intensional FOL is a mapping between the set \mathbb{L} of formulae of the logic language and intensional entities in \mathcal{D} , $I : \mathbb{L} \rightarrow \mathcal{D}$, is a kind of "conceptualization", such that an open-sentence (virtual predicate) $\phi(x_1, \dots, x_k)$ with a tuple \mathbf{x} of all free variables (x_1, \dots, x_k) is mapped into a k -ary *concept*, that is, an intensional entity $u = I(\phi(x_1, \dots, x_k)) \in D_k$, and (closed) sentence ψ into a proposition (i.e., *logic concept*) $v = I(\psi) \in D_0$ with $I(\top) = Truth \in D_0$ for a FOL tautology \top . This interpretation I is extended also to the terms (called as denotation as well). A language constant \bar{c} is mapped into a particular (an extensional entity) $a = I(\bar{c}) \in D_{-1}$

if it is a proper name, otherwise in a correspondent concept in \mathcal{D} . For each k -ary atom $r_i(\mathbf{x})$, $I(\langle r_i(\mathbf{x}) \rangle_{\mathbf{x}})$ is the relation-name (symbol) $r_i \in \mathbb{R}$ (only if r_i is not defined as a language constant as well). The extension of I to the complex abstracted terms is given in [14] (in Definition 4).

An assignment $g : \mathcal{V} \rightarrow \mathcal{D}$ for variables in \mathcal{V} is applied only to free variables in terms and formulae. Such an assignment $g \in \mathcal{D}^{\mathcal{V}}$ can be recursively uniquely extended into the assignment $g^* : \mathcal{TX} \rightarrow \mathcal{D}$, where \mathcal{TX} denotes the set of all terms with variables in $X \subseteq \mathcal{V}$ (here I is an intensional interpretation of this FOL, as explained in what follows), by :

1. $g^*(t_k) = g(x) \in \mathcal{D}$ if the term t_k is a variable $x \in \mathcal{V}$.
2. $g^*(t_k) = I(\bar{c}) \in \mathcal{D}$ if the term t_k is a constant \bar{c} .
3. if t_k is an abstracted term $\langle \phi \rangle_{\alpha}^{\beta}$, then $g^*(\langle \phi \rangle_{\alpha}^{\beta}) = I(\phi[\beta/g(\beta)]) \in D_k, k = |\alpha|$ (i.e., the number of variables in α), where $g(\beta) = g(y_1, \dots, y_m) = (g(y_1), \dots, g(y_m))$ and $[\beta/g(\beta)]$ is a uniform replacement of each i -th variable in the list β with the i -th constant in the list $g(\beta)$. Notice that α is the list of all free variables in the formula $\phi[\beta/g(\beta)]$.

We denote by t_k/g (or ϕ/g) the ground term (or formula) without free variables, obtained by assignment g from a term t_k (or a formula ϕ), and by $\phi[x/t_k]$ the formula obtained by uniformly replacing x by a term t_k in ϕ .

The distinction between intensions and extensions is important especially because we are now able to have an *equational theory* over intensional entities (as $\langle \phi \rangle$), that is predicate and function "names", that is separate from the extensional equality of relations and functions. An *extensionalization function* h assigns to the intensional elements of \mathcal{D} an appropriate extension as follows: for each proposition $u \in D_0$, $h(u) \in \{f, t\} \subseteq \mathcal{P}(D_{-1})$ is its extension (true or false value); for each n -ary concept $u \in D_n$, $h(u)$ is a subset of \mathcal{D}^n (n -th Cartesian product of \mathcal{D}); in the case of particulars $u \in D_{-1}$, $h(u) = u$.

We define $\mathcal{D}^0 = \{\langle \rangle\}$, so that $\{f, t\} = \mathcal{P}(\mathcal{D}^0)$, where \mathcal{P} is the powerset operator. Thus we have (we denote the disjoint union by '+'):

$$h = (h_{-1} + \sum_{i \geq 0} h_i) : \sum_{i \geq -1} D_i \longrightarrow D_{-1} + \sum_{i \geq 0} \mathcal{P}(D^i),$$

where $h_{-1} = id : D_{-1} \rightarrow D_{-1}$ is identity mapping, the mapping $h_0 : D_0 \rightarrow \{f, t\}$ assigns the truth values in $\{f, t\}$ to all propositions, and the mappings $h_i : D_i \rightarrow \mathcal{P}(D^i)$, $i \geq 1$, assign an extension to all concepts. Thus, the intensions can be seen as *names* of abstract or concrete entities, while the extensions correspond to various rules that these entities play in different worlds.

Remark: (Tarski's constraints) This intensional semantics has to preserve standard Tarski's semantics of the FOL. That is, for any formula $\phi \in \mathbb{L}$ with a tuple of free variables (x_1, \dots, x_k) , and $h \in \mathcal{E}$, the following conservative conditions for all assignments $g, g' \in \mathcal{D}^{\mathcal{V}}$ has to be satisfied:

$$(T) \quad h(I(\phi/g)) = t \quad \text{iff} \quad (g(x_1), \dots, g(x_k)) \in h(I(\phi));$$

and, if ϕ is a predicate letter p , $k = ar(p) \geq 2$ which represents a $(k-1)$ -ary functional symbol f^{k-1} in standard FOL,

$$(TF) \quad h(I(\phi/g)) = h(I(\phi/g')) = t \quad \text{and} \quad \forall_{1 \leq i \leq k-1} (g'(x_i) = g(x_i)) \quad \text{implies} \quad g'(x_{k+1}) = g(x_{k+1}).$$

□

Thus, intensional FOL has a simple Tarski's first-order semantics, with a decidable unification problem, but we need also the actual world mapping which maps any intensional entity to its *actual world extension*. In what follows we will identify a *possible world* by a particular mapping which assigns, in such a possible world, the extensions to intensional entities. This is direct bridge between an intensional FOL and a possible worlds representation [18,19,20,21,22,13], where the intension (meaning) of a proposition is a *function*, from a set of possible worlds \mathcal{W} into the set of truth-values. Consequently, \mathcal{E} denotes the set of possible *extensionalization functions* h satisfying the constraint (T). Each $h \in \mathcal{E}$ may be seen as a *possible world* (analogously to Montague's intensional semantics for natural language [20,22]), as it has been demonstrated in [23,24], and given by the bijection $is : \mathcal{W} \simeq \mathcal{E}$.

Now we are able to formally define this intensional semantics [13]:

Definition 3. TWO-STEP INTENSIONAL SEMANTICS:

Let $\mathfrak{R} = \bigcup_{k \in \mathbb{N}} \mathcal{P}(\mathcal{D}^k) = \sum_{k \in \mathbb{N}} \mathcal{P}(\mathcal{D}^k)$ be the set of all k -ary relations, where $k \in \mathbb{N} = \{0, 1, 2, \dots\}$. Notice that $\{f, t\} = \mathcal{P}(\mathcal{D}^0) \in \mathfrak{R}$, that is, the truth values are extensions in \mathfrak{R} . The intensional semantics of the logic language with the set of formulae L can be represented by the mapping

$$L \xrightarrow{I} \mathcal{D} \Rightarrow_{w \in \mathcal{W}} \mathfrak{R},$$

where \xrightarrow{I} is a fixed intensional interpretation $I : L \rightarrow \mathcal{D}$ and $\Rightarrow_{w \in \mathcal{W}}$ is the set of all extensionalization functions $h = is(w) : \mathcal{D} \rightarrow \mathfrak{R}$ in \mathcal{E} , where $is : \mathcal{W} \rightarrow \mathcal{E}$ is the mapping from the set of possible worlds to the set of extensionalization functions.

We define the mapping $I_n : L_{op} \rightarrow \mathfrak{R}^{\mathcal{W}}$, where L_{op} is the subset of formulae with free variables (virtual predicates), such that for any virtual predicate $\phi(x_1, \dots, x_k) \in L_{op}$ the mapping $I_n(\phi(x_1, \dots, x_k)) : \mathcal{W} \rightarrow \mathfrak{R}$ is the Montague's meaning (i.e., intension) of this virtual predicate [18,19,20,21,22], that is, the mapping which returns with the extension of this (virtual) predicate in each possible world $w \in \mathcal{W}$.

Another relevant question w.r.t. this two-step interpretations of an intensional semantics is how in it is managed the extensional identity relation \doteq (binary predicate of the identity) of the FOL. Here this extensional identity relation is mapped into the binary concept $Id = I(\doteq(x, y)) \in D_2$, such that $(\forall w \in \mathcal{W})(is(w)(Id) = R_{=})$, where $\doteq(x, y)$ (i.e., $p_1^2(x, y)$) denotes an atom of the FOL of the binary predicate for identity in FOL, usually written by FOL formula $x \doteq y$.

Note that here we prefer to distinguish this *formal symbol* $\doteq \in \mathbb{R}$ of the built-in identity binary predicate letter in the FOL, from the standard mathematical symbol '=' used in all mathematical definitions in this paper.

In what follows we will use the function $f_{<>} : \mathfrak{R} \rightarrow \mathfrak{R}$, such that for any relation $R \in \mathfrak{R}$, $f_{<>}(R) = \{<>\}$ if $R \neq \emptyset$; \emptyset otherwise. Let us define the following set of algebraic operators for relations in \mathfrak{R} :

1. binary operator $\bowtie_S : \mathfrak{R} \times \mathfrak{R} \rightarrow \mathfrak{R}$, such that for any two relations $R_1, R_2 \in \mathfrak{R}$, the $R_1 \bowtie_S R_2$ is equal to the relation obtained by natural join of these two relations if S is a non empty set of pairs of joined columns of respective relations (where the first argument is the column index of the relation R_1 while the second argument is the column index of the joined column of the relation R_2); otherwise it is

equal to the cartesian product $R_1 \times R_2$.

For example, the logic formula $\phi(x_i, x_j, x_k, x_l, x_m) \wedge \psi(x_l, y_i, x_j, y_j)$ will be traduced by the algebraic expression $R_1 \bowtie_S R_2$ where $R_1 \in \mathcal{P}(\mathcal{D}^5), R_2 \in \mathcal{P}(\mathcal{D}^4)$ are the extensions for a given Tarski's interpretation of the virtual predicate ϕ, ψ relatively, so that $S = \{(4, 1), (2, 3)\}$ and the resulting relation will have the following ordering of attributes: $(x_i, x_j, x_k, x_l, x_m, y_i, y_j)$.

2. unary operator $\sim: \mathfrak{R} \rightarrow \mathfrak{R}$, such that for any k-ary (with $k \geq 0$) relation $R \in \mathcal{P}(\mathcal{D}^k) \subset \mathfrak{R}$ we have that $\sim(R) = \mathcal{D}^k \setminus R \in \mathcal{D}^k$, where ' \setminus ' is the substraction of relations. For example, the logic formula $\neg\phi(x_i, x_j, x_k, x_l, x_m)$ will be traduced by the algebraic expression $\mathcal{D}^5 \setminus R$ where R is the extensions for a given Tarski's interpretation of the virtual predicate ϕ .
3. unary operator $\pi_{-m}: \mathfrak{R} \rightarrow \mathfrak{R}$, such that for any k-ary (with $k \geq 0$) relation $R \in \mathcal{P}(\mathcal{D}^k) \subset \mathfrak{R}$ we have that $\pi_{-m}(R)$ is equal to the relation obtained by elimination of the m-th column of the relation R if $1 \leq m \leq k$ and $k \geq 2$; equal to $f_{<>}(R)$ if $m = k = 1$; otherwise it is equal to R .

For example, the logic formula $(\exists x_k)\phi(x_i, x_j, x_k, x_l, x_m)$ will be traduced by the algebraic expression $\pi_{-3}(R)$ where R is the extensions for a given Tarski's interpretation of the virtual predicate ϕ and the resulting relation will have the following ordering of attributes: (x_i, x_j, x_l, x_m) .

Notice that the ordering of attributes of resulting relations corresponds to the method used for generating the ordering of variables in the tuples of free variables adopted for virtual predicates.

Definition 4. *Intensional algebra for the intensional FOL in Definition 2 is a structure $\mathcal{A}_{int} = (\mathcal{D}, f, t, Id, Truth, \{conj_S\}_{S \in \mathcal{P}(\mathbb{N}^2)}, neg, \{exists_n\}_{n \in \mathbb{N}})$, with binary operations $conj_S: D_I \times D_I \rightarrow D_I$, unary operation $neg: D_I \rightarrow D_I$, unary operations $exists_n: D_I \rightarrow D_I$, such that for any extensionalization function $h \in \mathcal{E}$, and $u \in D_k, v \in D_j, k, j \geq 0$,*

1. $h(Id) = R_=$ and $h(Truth) = \{<>\}$.
2. $h(conj_S(u, v)) = h(u) \bowtie_S h(v)$, where \bowtie_S is the natural join operation defined above and $conj_S(u, v) \in D_m$ where $m = k + j - |S|$ if for every pair $(i_1, i_2) \in S$ it holds that $1 \leq i_1 \leq k, 1 \leq i_2 \leq j$ (otherwise $conj_S(u, v) \in D_{k+j}$).
3. $h(neg(u)) = \sim(h(u)) = \mathcal{D}^k \setminus (h(u))$, where \sim is the operation defined above and $neg(u) \in D_k$.
4. $h(exists_n(u)) = \pi_{-n}(h(u))$, where π_{-n} is the operation defined above and $exists_n(u) \in D_{k-1}$ if $1 \leq n \leq k$ (otherwise $exists_n$ is the identity function).

Notice that for $u \in D_0$, $h(neg(u)) = \sim(h(u)) = \mathcal{D}^0 \setminus (h(u)) = \{<>\} \setminus (h(u)) \in \{f, t\}$.

Intensional interpretation $I: \mathbb{L} \rightarrow \mathcal{D}$ satisfies the following homomorphic extension:

1. The logic formula $\phi(x_i, x_j, x_k, x_l, x_m) \wedge \psi(x_l, y_i, x_j, y_j)$ will be intensionally interpreted by the concept $u_1 \in D_7$, obtained by the algebraic expression $conj_S(u, v)$ where $u = I(\phi(x_i, x_j, x_k, x_l, x_m)) \in D_5, v = I(\psi(x_l, y_i, x_j, y_j)) \in D_4$ are the concepts of the virtual predicates ϕ, ψ , relatively, and $S = \{(4, 1), (2, 3)\}$. Consequently, we have that for any two formulae $\phi, \psi \in \mathbb{L}$ and a particular operator $conj_S$ uniquely determined by tuples of free variables in these two formulae, $I(\phi \wedge \psi) = conj_S(I(\phi), I(\psi))$.

2. The logic formula $\neg\phi(x_i, x_j, x_k, x_l, x_m)$ will be intensionally interpreted by the concept $u_1 \in D_5$, obtained by the algebraic expression $neg(u)$ where $u = I(\phi(x_i, x_j, x_k, x_l, x_m)) \in D_5$ is the concept of the virtual predicate ϕ . Consequently, we have that for any formula $\phi \in \mathbb{L}$, $I(\neg\phi) = neg(I(\phi))$.
3. The logic formula $(\exists x_k)\phi(x_i, x_j, x_k, x_l, x_m)$ will be intensionally interpreted by the concept $u_1 \in D_4$, obtained by the algebraic expression $exists_3(u)$ where $u = I(\phi(x_i, x_j, x_k, x_l, x_m)) \in D_5$ is the concept of the virtual predicate ϕ . Consequently, we have that for any formula $\phi \in \mathbb{L}$ and a particular operator $exists_n$ uniquely determined by the position of the existentially quantified variable in the tuple of free variables in ϕ (otherwise $n = 0$ if this quantified variable is not a free variable in ϕ), $I((\exists x)\phi) = exists_n(I(\phi))$.

Once one has found a method for specifying the interpretations of singular terms of \mathbb{L} (take in consideration the particularity of abstracted terms), the Tarski-style definitions of truth and validity for \mathbb{L} may be given in the customary way. What is proposed specifically in [14] is a method for characterizing the intensional interpretations of singular terms of \mathbb{L} in such a way that a given singular abstracted term $\langle\phi\rangle_\alpha^\beta$ will denote an appropriate property, relation, or proposition, depending on the value of $m = |\alpha|$.

Notice that if $\beta = \emptyset$ is the empty list, then $I(\langle\phi\rangle_\alpha^\beta) = I(\phi)$. Consequently, the denotation of $\langle\phi\rangle$ is equal to the meaning of a proposition ϕ , that is, $I(\langle\phi\rangle) = I(\phi) \in D_0$. In the case when ϕ is an atom $p_i(x_1, \dots, x_m)$ then $I(\langle p_i(x_1, \dots, x_m) \rangle_{x_1, \dots, x_m}^\beta) = I(p_i(x_1, \dots, x_m)) \in D_m$, while $I(\langle p_i(x_1, \dots, x_m) \rangle_{x_1, \dots, x_m}^{x_1, \dots, x_m}) = union(\{I(p_i(g(x_1), \dots, g(x_m))) \mid g \in \mathcal{D}^{x_1, \dots, x_m}\}) \in D_0$, with $h(I(\langle p_i(x_1, \dots, x_m) \rangle_{x_1, \dots, x_m}^{x_1, \dots, x_m})) = h(I((\exists x_1) \dots (\exists x_m) p_i(x_1, \dots, x_m))) \in \{f, t\}$. For example,

$$h(I(\langle p_i(x_1) \wedge \neg p_i(x_1) \rangle_{x_1}^{x_1})) = h(I((\exists x_1)(\langle p_i(x_1) \wedge \neg p_i(x_1) \rangle_{x_1}^{x_1}))) = f.$$

The interpretation of a more complex abstract $\langle\phi\rangle_\alpha^\beta$ is defined in terms of the interpretations of the relevant syntactically simpler expressions, because the interpretation of more complex formulae is defined in terms of the interpretation of the relevant syntactically simpler formulae, based on the intensional algebra above. For example, $I(p_i(x) \wedge p_k(x)) = conj_{\{(1,1)\}}(I(p_i(x)), I(p_k(x)))$, $I(\neg\phi) = neg(I(\phi))$, $I(\exists x_i)\phi(x_i, x_j, x_i, x_k) = exists_3(I(\phi))$.

Consequently, based on the intensional algebra in Definition 4 and on intensional interpretations of abstracted terms, it holds that the interpretation of any formula in \mathbb{L} (and any abstracted term) will be reduced to an algebraic expression over interpretations of primitive atoms in \mathbb{L} . This obtained expression is finite for any finite formula (or abstracted term), and represents the *meaning* of such finite formula (or abstracted term). Let $\mathcal{A}_{FOL} = (\mathbb{L}, \doteq, \top, \wedge, \neg, \exists)$ be a free syntax algebra for "First-order logic with identity \doteq ", with the set \mathbb{L} of first-order logic formulae, with \top denoting the tautology formula (the contradiction formula is denoted by $\neg\top$), with the set of variables in \mathcal{V} and the domain of values in \mathcal{D} .

Let us define the extensional relational algebra for the FOL by,

$$\mathcal{A}_{\mathfrak{R}} = (\mathfrak{R}, R_-, \{<>\}, \{\bowtie_S\}_{S \in \mathcal{P}(\mathbb{N}^2)}, \sim, \{\pi_{-n}\}_{n \in \mathbb{N}}),$$

where $\{<>\} \in \mathfrak{R}$ is the algebraic value correspondent to the logic truth, and R_- is the binary relation for extensionally equal elements. We use '=' for the extensional identity for relations in \mathfrak{R} .

Then, for any Tarski's interpretation I_T its unique extension to all formulae $I_T^* : \mathbb{L} \rightarrow \mathfrak{R}$ is also the homomorphism $I_T^* : \mathcal{A}_{FOL} \rightarrow \mathcal{A}_{\mathfrak{R}}$ from the free syntax FOL algebra into this extensional relational algebra.

Consequently, we obtain the following Intensional/extensional FOL semantics [13]:
For any Tarski's interpretation I_T of the FOL, the following diagram of homomorphisms commutes,

$$\begin{array}{ccc}
 & \mathcal{A}_{int} \text{ (concepts/meaning)} & \\
 \text{intensional interpret. } I \nearrow & \xrightarrow{\text{Frege/Russell semantics}} & \searrow h \text{ (extensionalization)} \\
 \mathcal{A}_{FOL} \text{ (syntax)} & \xrightarrow{I_T^* \text{ (Tarski's interpretation)}} & \mathcal{A}_{\mathfrak{R}} \text{ (denotation)}
 \end{array}$$

where $h = is(w)$ where $w = I_T \in \mathcal{W}$ is the explicit possible world (extensional Tarski's interpretation).

This homomorphic diagram formally express the fusion of Frege's and Russell's semantics [25,26,27] of meaning and denotation of the FOL language, and renders mathematically correct the definition of what we call an "intuitive notion of intensionality", in terms of which a language is intensional if denotation is distinguished from sense: that is, if both a denotation and sense is ascribed to its expressions. This notion is simply adopted from Frege's contribution (without its infinite sense-hierarchy, avoided by Russell's approach where there is only one meaning relation, one fundamental relation between words and things, here represented by one fixed intensional interpretation I), where the sense contains mode of presentation (here described algebraically as an algebra of concepts (intensions) \mathcal{A}_{int} , and where sense determines denotation for any given extensionalization function h (correspondent to a given Traski's interpretaion I_T). More about the relationships between Frege's and Russell's theories of meaning may be found in the Chapter 7, "Extensionality and Meaning", in [28].

As noted by Gottlob Frege and Rudolf Carnap (he uses terms Intension/extension in the place of Frege's terms sense/denotation [29]), the two logic formulae with the same denotation (i.e., the same extension for a given Tarski's interpretation I_T) need not have the same sense (intension), thus such co-denotational expressions are not *substitutable* in general.

In fact there is exactly *one* sense (meaning) of a given logic formula in \mathbb{L} , defined by the uniquely fixed intensional interpretation I , and *a set* of possible denotations (extensions) each determined by a given Tarski's interpretation of the FOL as follows from Definition 3,

$$\mathbb{L} \xrightarrow{I} \mathcal{D} \Longrightarrow_{h=is(I_T), I_T \in \mathcal{W}} \mathfrak{R}.$$

Often 'intension' has been used exclusively in connection with possible worlds semantics, however, here we use (as many others; as Bealer for example) 'intension' in a more wide sense, that is as an *algebraic expression* in the intensional algebra of meanings (concepts) \mathcal{A}_{int} which represents the structural composition of more complex concepts (meanings) from the given set of atomic meanings. Consequently, not only the denotation (extension) is compositional, but also the meaning (intension) is compositional.

4 Canonical models for IRDBs

The application of the intensional FOL semantics to the Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ in Definition 1 with the user defined RDB schema $\mathcal{A} = (S_A, \Sigma_A)$ and the vector big table r_V can be summarized in what follows:

- Each relational name (symbol) $r_k \in S_A = \{r_1, \dots, r_n\}$ with the arity $m = ar(r_k)$, is an intensional m-ary concept, so that $r_k = I(\langle r_k(\mathbf{x}) \rangle_{\mathbf{x}}) \in D_m$, for a tuple of variables $\mathbf{x} = \langle x_1, \dots, x_m \rangle$ and any intensional interpretation I .
For a given Tarski's interpretation I_T , the extensionalization function h is determined by $h(r_k) = \|r_k\| = \{\langle d_1, \dots, d_m \rangle \in \mathcal{D}^m \mid I_T(r_k(d_1, \dots, d_m)) = t\} = I_T(r_k) \in \mathcal{A}$. The instance database A of the user-defined RDB schema \mathcal{A} is a model of \mathcal{A} if it satisfies all integrity constraints in Σ_A .
- The relational symbol (name) r_V of the vector big table is a particular (extensional entity), $r_V \in D_{-1}$, so that $h(r_V) = r_V$ (the name of the database \mathcal{A}). For a given model $A = \{\|r_1\|, \dots, \|r_n\|\}$ of the user-defined RDB schema \mathcal{A} , correspondent to a given Tarski's interpretation I_T , its extension is determined by $I_T(r_V) = \|r_V\| = \vec{A}$.
- Intensional nature of the IRDB is evident in the fact that each tuple $\langle r_k, Hash(d_1, \dots, d_m), nr_{r_k}(i), d_i \rangle \in \vec{A}$, corresponding to the atom $r_V(y_1, y_2, y_3, y_4)/g$ for an assignment g such that $g(y_1) = r_k \in D_m, g(y_3) = nr_{r_k}(i) \in D_{-1}, g(y_2) = Hash(d_1, \dots, d_m) \in D_{-1}$ and $g(y_4) = d_i \in \mathcal{D}$, is equal to the intensional tuple $\langle I(\langle r_k(\mathbf{x}) \rangle_{\mathbf{x}}), Hash(d_1, \dots, d_m), nr_{r_k}(i), d_i \rangle$.
Notice that the intensional tuples are different from ordinary tuples composed by only particulars (extensional elements) in D_{-1} , what is the characteristics of the standard FOL (where the domain of values is equal to D_{-1}), while here the "value" $r_k = I(\langle r_k(\mathbf{x}) \rangle_{\mathbf{x}}) \in D_m$ is an m-ary intensional concept, for which $h(r_k) \neq r_k$ is an m-ary relation (while for all ordinary values $d \in D_{-1}$, $h(d) = d$).

The intensional Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ in Definition 1 is used in the way that the global schema is only virtual (empty) database with a user-defined schema $\mathcal{A} = (S_A, \Sigma_A)$ used to define the SQL user-defined query which then has to be equivalently rewritten over the vector relation r_V in order to obtain the answer to this query. Thus, the information of the database is stored only in the big table $\|r_V\|$. Thus, the materialization of the original user-defined schema \mathcal{A} can be obtained by the following operation:

Definition 5. MATERIALIZATION OF THE RDB

Given a user-defined RDB schema $\mathcal{A} = (S_A, \Sigma_A)$ with $S_A = \{r_1, \dots, r_n\}$ and a big vector table $\|r_V\|$, the non SQL operation **MATTER** which materializes the schema \mathcal{A} into its instance database $A = \{R_1, \dots, R_n\}$ where $R_k = \|r_k\|$, for $k = 1, \dots, n$, is given by the following mapping, for any $R \subseteq \|r_V\|$:

$$(r_k, R) \mapsto \{ \langle v_1, \dots, v_{ar(r_k)} \rangle \mid \exists y \in \pi_2(R) ((r_V(r_k, y, nr_{r_k}(1), v_1) \underline{\vee} v_1 \text{NULL}) \wedge \dots \wedge (r_V(r_k, y, nr_{r_k}(ar(r_k)), v_{ar(r_k)}) \underline{\vee} v_{ar(r_k)} \text{NULL})) \},$$

so that the materialization of the schema \mathcal{A} is defined by

$$R_k = \|r_k\| \triangleq \text{MATTER}(r_k, \|r_V\|) \text{ for each } r_k \in S_A.$$

The canonical models of the intensional Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ in Definition 1 are the instances A of the schema \mathcal{A} such that

$\|r_k\| = \text{MATTER}(r_k, \bigcup_{v \in \|r_k\|} \text{PARSE}(r_k, v))$, that is, when

$$A = \{\text{MATTER}(r_k, \vec{A}) \mid r_k \in S_A\}.$$

The canonical models of such intensional Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ can be provided in a usual logical framework as well:

Proposition 1 *Let the IRDB be given by a Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ for a used-defined global schema $\mathcal{A} = (S_A, \Sigma_A)$ with $S_A = \{r_1, \dots, r_n\}$, the source schema $\mathcal{S} = (\{r_V\}, \emptyset)$ with the vector big data relation r_V and the set of mapping tgds \mathcal{M} from the source schema into the relations of the global schema. Then a canonical model of \mathcal{I} is any model of the schema $\mathcal{A}^+ = (S_A \cup \{r_V\}, \Sigma_A \cup \mathcal{M} \cup \mathcal{M}^{OP})$, where \mathcal{M}^{OP} is an opposite mapping tgds from \mathcal{A} into r_V given by the following set of tgds: $\mathcal{M}^{OP} = \{\forall x_1, \dots, x_{ar(r_k)} ((r_k(x_1, \dots, x_{ar(r_k)}) \wedge x_i \text{NOT NULL}) \Rightarrow r_V(r_k, Hash(x_1, \dots, x_{ar(r_k)}), nr_{r_k}(i), x_i)) \mid 1 \leq i \leq ar(r_k), r_k \in S_A\}$.*

Proof: It is enough to show that for each $r_k \in S_A$, and $\mathbf{x} = (x_1, \dots, x_{ar(r_k)})$,

$$r_k(\mathbf{x}) \Leftrightarrow ((r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(1), x_1) \underline{\vee} x_1 \text{NULL}) \wedge \dots \wedge (r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(ar(r_k)), x_{ar(r_k)}) \underline{\vee} x_{ar(r_k)} \text{NULL})).$$

From \mathcal{M}^{OP} we have

$$\neg r_k(\mathbf{x}) \vee \neg x_i \text{NOT NULL} \vee r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i), \text{ that is,}$$

$$(a) \quad \neg r_k(\mathbf{x}) \vee (x_i \text{NULL} \vee r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i)).$$

From the other side, from the fact that we have the constraint NOT NULL for the attribute value (in Definition 1), then

$$\neg r_k(\mathbf{x}) \vee \neg x_i \text{NULL} \vee \neg r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i), \text{ that is}$$

$$(b) \quad \neg r_k(\mathbf{x}) \vee \neg (x_i \text{NULL} \wedge r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i)),$$

is true and also the conjunction of (a) and (b) has to be true, i.e.,

$$(\neg r_k(\mathbf{x}) \vee (x_i \text{NULL} \vee r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i))) \wedge (\neg r_k(\mathbf{x}) \vee \neg (x_i \text{NULL} \wedge r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i))), \text{ thus, by distributivity,}$$

$$(b) \quad \neg r_k(\mathbf{x}) \vee (x_i \text{NULL} \underline{\vee} r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(i), x_i)).$$

If we repeat this for all $1 \leq i \leq ar(r_k)$ and conjugate all these true formula, again by distributive property of conjunction \wedge , we obtain

$$\neg r_k(\mathbf{x}) \vee ((x_1 \text{NULL} \underline{\vee} r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(1), x_1)) \wedge \dots$$

$$\wedge (x_{ar(r_k)} \text{NULL} \underline{\vee} r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(ar(r_k)), x_{ar(r_k)}))), \text{ that is,}$$

$$(c) \quad r_k(\mathbf{x}) \Rightarrow ((r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(1), x_1) \underline{\vee} x_1 \text{NULL}) \wedge \dots$$

$$\wedge (r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(ar(r_k)), x_{ar(r_k)}) \underline{\vee} x_{ar(r_k)} \text{NULL})).$$

Moreover, from Definition we also have

$$(d) \quad r_k(\mathbf{x}) \Leftarrow ((r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(1), x_1) \underline{\vee} x_1 \text{NULL}) \wedge \dots$$

$$\wedge (r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(ar(r_k)), x_{ar(r_k)}) \underline{\vee} x_{ar(r_k)} \text{NULL})),$$

That is, the logical equivalence of the formula on the left and on the right side of the logical implication, and hence if the atom $r_k(\mathbf{x})$ is true for some assignment to the variables g so that the tuple $\langle g(x_1), \dots, g(x_{ar(r_k)}) \rangle$ is in relation $r_k \in \mathcal{A}$, then for the same assignment g the every formula

$$r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(1), x_1) \underline{\vee} x_1 \text{NULL},$$

...

$$r_V(r_k, Hash(\mathbf{x}), nr_{r_k}(ar(r_k)), x_{ar(r_k)}) \underline{\vee} x_{ar(r_k)} \text{NULL}$$

has to be true and hence generates the tuples in r_V for NOT NULL values of $g(x_i)$, $1 \leq i \leq ar(r_k)$.

Notice that the implication (c) corresponds to the nonSQL operation PARSE, while the implication (d) is the logical semantics of the non SQL operation MATTER.

Consequently, we obtain $\|r_k\| = \underline{\text{MATTER}}(r_k, \bigcup_{v \in \|r_k\|} \underline{\text{PARSE}}(r_k, v))$, that is, $A = \{\underline{\text{MATTER}}(r_k, \vec{A}) \mid r_k \in S_A\}$ and the database instance A which satisfies all integrity constraints $\Sigma_A \cup \mathcal{M} \cup \mathcal{M}^{OP}$ is the canonical model of the intensional Data Integration system $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$.

□

By joking with the words, we can say that "by PARSEing the MATTER we obtain the pure energy" of the big vector relation, and conversely, "by condensing the PARSEd energy we obtain the common MATTER" in a standard RDB.

The fact is that we do not need both of them because they are equivalent, so instead of the more (schema) rigid RDB matter in \mathcal{A} we prefer to use the non rigid pure energy of the big vector table r_V . But we are also able to render more flexible this approach and to decide only a subset of relations to be the intensional concepts whose extension has to be parsed in to the vector big table r_V . For standard legacy systems we can chose to avoid at all to have the intensional concepts, thus to have the standard RDBs with standard FOL Tarski's semantics. By declaring any of the relational names $r_k \in S_A$ as an intensional concept, we conservatively extend the Tarski's semantics for the FOL in order to obtain a more expressive intensional FOL semantics for the IRDBs.

The fact that we assumed r_V to be only a particular (extensional entity) is based on the fact that it always will be materialized (thus non empty relational table) as standard tables in the RDBs. The other reason is that the extension $h(r_V)$ has not to be equal to the vector relation (the set of tuples) $\|r_V\|$ but to the *set of relations* in the instance database A . Consequently, we do not use the r_V (equal to the name of the database \mathcal{A}) as a value in the tuples of other relations and we do not use the parsing used for all relations in the user-defined RDB schema \mathcal{A} assumed to be the intensional concepts as well.

If we would decide to use also r_V as an intensional concept in D_4 we would be able to parse it as all other intensional concepts in \mathcal{A} into itself, and such recursive definition will render (only theoretically) an infinite extension of the r_V , thus non applicable, as follows. Let $r_k = \text{Person} \in \mathcal{A}$ be an user-defined relational table, and $Pname$ an attribute of this table, and let ID be the t-index value obtained by Hash function from one tuple of r_k where the value of the attribute $Pname$ is "Marco Aurelio", then we will have this tuples in the vector table with (database) name $r_V \in D_4$:

1. $\langle \text{Person}, ID, Pname, \text{MarcoAurelio} \rangle \in \|r_V\|$;

then by parsing this tuple 1, we will obtain for $ID_1 = \text{Hash}(\text{Person}, ID, Pname, \text{MarcoAurelio})$ the following new tuples

2. $\langle r_V, ID_1, r\text{-name}, \text{Person} \rangle \in \|r_V\|$;

3. $\langle r_V, ID_1, t\text{-index}, ID \rangle \in \|r_V\|$;

4. $\langle r_V, ID_1, a\text{-name}, Pname \rangle \in \|r_V\|$;

5. $\langle r_V, ID_1, value, \text{MarcoAurelio} \rangle \in \|r_V\|$;

then by parsing this tuple 2, we will obtain for $ID_2 = \text{Hash}(r_V, ID_1, r\text{-name}, \text{Person})$ the following new tuples

6. $\langle r_V, ID_2, \text{r-name}, \text{Vector} \rangle \in \|r_V\|;$
 7. $\langle r_V, ID_2, \text{t-index}, ID_1 \rangle \in \|r_V\|;$
 8. $\langle r_V, ID_2, \text{a-name}, \text{r-name} \rangle \in \|r_V\|;$
 9. $\langle r_V, ID_2, \text{value}, \text{Person} \rangle \in \|r_V\|;$
- then by parsing this tuple 2, we will obtain for $ID_3 = \text{Hash}(r_V, ID_2, \text{r-name}, \text{Person})$ the following new tuples
10. $\langle r_V, ID_3, \text{r-name}, r_V \rangle \in \|r_V\|;$

...

Thus, as we see, the tuple 10 is equal to the tuple 6, but only with new t-index (tuple index) value, so by continuing this process, theoretically (if we do not pose the limits for the values of t-indexes) we obtain an infinite process and an infinite extension of r_V . Obviously, it can not happen in real RDBs, because the length of the attribute t-index is finite so that at some point we will obtain the previously generated value for this attribute (we reach a fixed point), and from the fact that this attribute is a part of the primary key this tuple would not be inserted in r_V because r_V contains the same tuple already.

Notice that this process is analogous to the selfreferencing process, where we try to use an intensional concept *as an element of itself* that has to be avoided and hence there is no sense to render r_V an intensional concept. Consequently, the IRDB has at least one relational table which is not an intensional concept and which will not be parsed: the vector big table, which has this singular built-in property in every IRDB.

5 NewSQL property of the IRDBs

This last section we will dedicate to demonstrate that the IRDBs are complete w.r.t. the standard SQL. This demonstration is based on the fact that each SQL query, defined over the user-defined schema \mathcal{A} , which (in full intensional immersion) is composed by the intensional concepts, will be executed over standard relational tables that *are not* the intensional concepts. If a query is defined over the non-intensional concepts (relations) in \mathcal{A} in this case it will be directly executed over these relational tables as in every RDB. If a query is defined over the intensional concepts in \mathcal{A} (which will remain *empty tables*, i.e., non materialized) then we need to demonstrate the existence of an effective query-rewriting into an equivalent SQL query over the (non-intensional concept) vector big table r_V . In order to define this query-rewriting, we will shortly introduce the abstract syntax and semantics of Codd's relational algebra, as follows.

Five primitive operators of Codd's algebra are: the selection, the projection, the Cartesian product (also called the cross-product or cross-join), the set union, and the set difference. Another operator, rename, was not noted by Codd, but the need for it is shown by the inventors of Information Systems Base Language (ISBL) for one of the earliest database management systems which implemented Codd's relational model of data. These six operators are fundamental in the sense that if we omit any one of them, we will lose expressive power. Many other operators have been defined in terms of these six. Among the most important are set intersection, division, and the natural join. In fact, ISBL made a compelling case for replacing the Cartesian product with the natural join, of which the Cartesian product is a degenerate case.

Recall that two relations r_1 and r_2 are union-compatible iff $\{atr(r_1)\} = \{atr(r_2)\}$, where for a given list (or tuple) of the attributes $\mathbf{a} = atr(r) = \langle a_1, \dots, a_k \rangle = \langle atr_r(1), \dots, atr_r(k) \rangle$, we denote the set $\{a_1, \dots, a_k\}$ by $\{atr(r)\}$, $k = ar(r)$, with the injective function $nr_r : \{1, \dots, k\} \rightarrow SN$ which assigns distinct names to each column of this relation. If a relation r_2 is obtained from a given relation r_1 by permutating its columns, then we tell that they are not equal (in set theoretic sense) but that they are equivalent. Notice that in the RDB theory the two equivalent relations are considered equal as well. In what follows, given any two lists (tuples), $\mathbf{d} = \langle d_1, \dots, d_k \rangle$ and $\mathbf{b} = \langle b_1, \dots, b_m \rangle$ their concatenation $\langle d_1, \dots, d_k, b_1, \dots, b_m \rangle$ is denoted by $\mathbf{d} \& \mathbf{b}$, where ' $\&$ ' is the symbol for concatenation of the lists. By $\|r\|$ we denote the extension of a given relation (relational symbol) r ; it is extended to any term t_R of Codd's algebra, so that $\|t_R\|$ is the relation obtained by computation of this term. Let us briefly define these basic operators, and their correspondence with the formulae of FOL:

1. Rename is a unary operation written as $_ \text{RENAME } name_1 \text{ AS } name_2$ where the result is identical to input argument (relation) r except that the column i with name $nr_r(i) = name_1$ in all tuples is renamed to $nr_r(i) = name_2$. This operation is neutral w.r.t. the logic, where we are using the variables for the columns of relational tables and not their names.

2. Cartesian product is a binary operation $_ \text{TIMES } _$, written also as $_ \otimes _$, such that for the relations r_1 and r_2 , first we do the rename normalization of r_2 (w.r.t. r_1), denoted by r_2^ρ , such that:

For each k -th copy of the attribute a_i (or, equivalently, $a_i(0)$) of the m -th column of r_2 (with $1 \leq m \leq ar(r_2)$), denoted by $a_i(k) = atr_{r_2}(m) \in atr(r_2)$, such that the maximum index of the same attribute a_i in r_1 is $a_i(n)$, we change r_2 by:

1. $a_i(k) \mapsto a_i(k + n)$;
2. if $name_1 = nr_{r_2}(m)$ is a name that exists in the set of the column names in r_1 , then we change the naming function $nr_{r_2} : \{1, \dots, ar(r_2)\} \rightarrow SN$, by $nr_{r_2}(m) = name_2$, where $name_2 \in SN$ is a new name distinct from all other used names, and we define the renaming normalization ρ by mapping $name_1 \mapsto name_2$. The relation obtained from r_2 , after this renaming normalization, will be denoted by r_2^ρ . Then we define the new relation r (when both $\|r_1\| \neq \{\langle \rangle\}$ and $\|r_2\| \neq \{\langle \rangle\}$, i.e., when are not empty relations) by $r_1 \otimes r_2^\rho$, with $\|r\| \triangleq \{\mathbf{d}_1 \& \mathbf{d}_2 \mid \mathbf{d}_1 \in \|r_1\|, \mathbf{d}_2 \in \|r_2\|\}$, with the naming function $nr_r : \{1, \dots, ar(r_1) + ar(r_2)\} \rightarrow SN$, such that $nr_r(i) = nr_{r_1}(i)$ for $1 \leq i \leq ar(r_1)$ and $nr_r(i) = nr_{r_2}(i)$ for $1 + ar(r_1) \leq i \leq ar(r_1) + ar(r_2)$, and $atr_r : \{1, \dots, ar(r_1) + ar(r_2)\} \rightarrow \mathbf{att}$ function defined by $atr_r(i) = atr_{r_1}(i)$ for $1 \leq i \leq ar(r_1)$ and $atr_r(i) = atr_{r_2}(i)$ for $1 + ar(r_1) \leq i \leq ar(r_1) + ar(r_2)$.

This Cartesian product is given by the following logical equivalence, by considering the relational symbols as predicates,

$r(x_1, \dots, x_{ar(r_1)}, y_1, \dots, y_{ar(r_2)}) \Leftrightarrow (r_1(x_1, \dots, x_{ar(r_1)}) \wedge r_2(y_1, \dots, y_{ar(r_2)}))$, so that $\|r\| = \|r_1(x_1, \dots, x_{ar(r_1)}) \wedge r_2(y_1, \dots, y_{ar(r_2)})\|$.

(if $\|r_1\|$ is empty then $r_1 \otimes r_2^\rho = r_2$; if $\|r_2\|$ is empty then $r_1 \otimes r_2^\rho = r_1$).

3. Projection is a unary operation written as $_ [S]$, where S is a tuple of column names such that for a relation r_1 and $S = \langle nr_{r_1}(i_1), \dots, nr_{r_1}(i_k) \rangle$, with $k \geq 1$ and $1 \leq i_m \leq ar(r_1)$ for $1 \leq m \leq k$, and $i_m \neq i_j$ if $m \neq j$, we define the relation r

by: $r_1[S]$,

with $\|r\| = \|r_1\|$ if $\exists \text{name} \in S.\text{name} \notin nr(r_1)$; otherwise $\|r\| = \pi_{\langle i_1, \dots, i_k \rangle}(\|r_1\|)$,
 where $nr_r(m) = nr_{r_1}(i_m)$, $atr_r(m) = atr_{r_1}(i_m)$, for $1 \leq m \leq k$.

This projection is given by the following logical equivalence

$$r(x_{i_1}, \dots, x_{i_k}) \Leftrightarrow \exists x_{j_1} \dots x_{j_n} r_1(x_1, \dots, x_{ar(r_1)}),$$

where $n = ar(r_1) - k$ and for all $1 \leq m \leq n$, $j_m \notin \{i_1, \dots, i_k\}$, so that

$$\|r\| = \|\exists x_{j_1} \dots x_{j_n} r_1(x_1, \dots, x_{ar(r_1)})\|.$$

4. Selection is a unary operation written as $_ \text{WHERE } C$, where a condition C is a finite-length logical formula that consists of atoms $'(name_i \theta name_j)'$ or $'(name_i \theta \bar{d})'$,
 with built-in predicates $\theta \in \Sigma_\theta \supseteq \{=, >, <\}$, a constant \bar{d} , and the logical operators \wedge (AND), \vee (OR) and \neg (NOT), such that for a relation r_1 and $name_i, name_j$ the names of its columns, we define the relation r by

$r_1 \text{ WHERE } C$,

as the relation with $atr(r) = atr(r_1)$ and the function nr_r equal to nr_{r_1} , where $\|r\|$ is composed by the tuples in $\|r_1\|$ for which C is satisfied.

This selection is given by the following logical equivalence:

$$r(x_{i_1}, \dots, x_{i_k}) \Leftrightarrow (r_1(x_1, \dots, x_{ar(r_1)}) \wedge C(\mathbf{x})),$$

where $C(\mathbf{x})$ is obtained by substitution of each $name_i = nr_{r_1}(j)$ (of the j -th column of r_1) in the formula C by the variable x_j , so that

$$\|r\| = \|r_1(x_1, \dots, x_{ar(r_1)}) \wedge C(\mathbf{x})\|.$$

4.1 We assume as an *identity unary operation*, the operation $_ \text{WHERE } C$ when C is the atomic condition $\bar{1} \doteq \bar{1}$ (i.e., a tautology).

5. Union is a binary operation written as $_ \text{UNION } _$, such that for two union-compatible relations r_1 and r_2 , we define the relation r by: $r_1 \text{ UNION } r_2$,
 where $\|r\| \triangleq \|r_1\| \cup \|r_2\|$, with $atr(r) = atr(r_1)$, and the functions $atr_r = atr_{r_1}$, and $nr_r = nr_{r_1}$. This union is given by the following logical equivalence:

$$r(x_1, \dots, x_n) \Leftrightarrow (r_1(x_1, \dots, x_n) \vee r_2(x_1, \dots, x_n)),$$

where $n = ar(r) = ar(r_1) = ar(r_2)$, so that

$$\|r\| = \|r_1(x_1, \dots, x_n) \vee r_2(x_1, \dots, x_n)\|.$$

6. Set difference is a binary operation written as $_ \text{MINUS } _$ such that for two union-compatible relations r_1 and r_2 , we define the relation r by: $r_1 \text{ MINUS } r_2$,
 where $\|r\| \triangleq \{\mathbf{t} \mid \mathbf{t} \in \|r_1\| \text{ such that } \mathbf{t} \notin \|r_2\|\}$, with $atr(r) = atr(r_1)$, and the functions $atr_r = atr_{r_1}$, and $nr_r = nr_{r_1}$.

Let r_1 and r_2 be the predicates (relational symbols) for these two relations. Then their difference is given by the following logical equivalence:

$$r(x_1, \dots, x_n) \Leftrightarrow (r_1(x_1, \dots, x_n) \wedge \neg r_2(x_1, \dots, x_n)),$$

where $n = ar(r) = ar(r_1) = ar(r_2)$ and hence

$$\|r\| = \|r_1(x_1, \dots, x_n) \wedge \neg r_2(x_1, \dots, x_n)\|.$$

Natural join \bowtie_S is a binary operator, written as $(r_1 \bowtie_S r_2)$, where r_1 and r_2 are the relations. The result of the natural join is the set of all combinations of tuples in r_1 and r_2 that are equal on their common attribute names. In fact, $(r_1 \bowtie_S r_2)$ can be obtained by creating the Cartesian product $r_1 \otimes r_2$ and then by execution of the Selection with the condition C defined as a conjunction of atomic formulae $(nr_{r_1}(i) = nr_{r_2}(j))$ with $(nr_{r_1}(i), nr_{r_2}(j)) \in S$ (where i and j are the columns of the same attribute in r_1 and r_2 , respectively, i.e., satisfying $atr_{r_1}(i) = atr_{r_2}(j)$) that represents the equality of the

common attribute names of r_1 and r_2 . The natural join is arguably one of the most important operators since it is the relational counterpart of logical AND. Note carefully that if the same variable appears in each of two predicates that are linked by AND, then that variable stands for the same thing and both appearances must always be substituted by the same value. In particular, natural join allows the combination of relations that are associated by a foreign key. It can also be used to define composition of binary relations.

Altogether, the operators of relational algebra have identical expressive power to that of domain relational calculus or tuple relational calculus. However, relational algebra is less expressive than first-order predicate calculus without function symbols. Relational algebra corresponds to a *subset* of FOL (denominated *relational calculus*), namely Horn clauses without recursion and negation (or union of conjunctive queries). Consequently, relational algebra is essentially equivalent in expressive power to *relational calculus* (and thus FOL and queries defined in Section 2); this result is known as Codd's theorem. However, the negation, applied to a formula of the calculus, constructs a formula that may be true on an infinite set of possible tuples. To overcome this difficulty, Codd restricted the operands of relational algebra to finite relations only and also proposed restricted support for negation \neg (NOT) and disjunction \vee (OR). Codd defined the term "relational completeness" to refer to a language that is complete with respect to first-order predicate calculus apart from the restrictions he proposed. In practice, the restrictions have no adverse effect on the applicability of his relational algebra for database purposes.

Several papers have proposed new operators of an algebraic nature as candidates for addition to the original set. We choose the additional unary operator 'EXTEND. ADD $a, name$ AS e ' denoted shortly as $_ \langle a, name, e \rangle$, where a is a new added attribute as the new column (at the end of relation) with a new fresh name $name$ and e is an expression (in the most simple cases it can be the value NULL or a constant \bar{d} , or the i -th column name $nr(i)$ of the argument (i.e., relation) of this operation), for *update* relational algebra operators, in order to cover all of the basic features of data manipulation (DML) aspects of a relation models of data, so that

- We define a unary operator $_ \langle a, name, e \rangle$, for an attribute $a \in \mathbf{att}$, its name, and expression e , as a function with a set of column names, such that for a relation r_1 and expression e composed of the names of the columns of r_1 with $n = ar(r_1)$, we obtain the $(ar(r_1) + 1)$ -ary relation r by $_ \langle a, name, e \rangle(r_1)$, with naming function $nr_r : \{ar(r_1) + 1\} \rightarrow SN$ such that $nr_r(i) = nr_{r_1}(i)$ if $i \leq ar(r_1)$; $nr_r(ar(r_1) + 1) = name$ otherwise, being a fresh new name for this column; with the attribute function $atr_r : \{ar(r_1) + 1\} \rightarrow \mathbf{att}$ such that $atr_r(i) = atr_{r_1}(i)$ if $i \leq ar(r_1)$; $atr_r(ar(r_1) + 1) = a$ otherwise, and $\|r\| = \{ \langle \rangle \} \cup \{ \mathbf{d} \& e(\mathbf{d}) \mid \mathbf{d} \in \|r_1\| \}$, where $e(\mathbf{d}) \in dom(a)$ is a constant or the value obtained from the function e where each name $nr_r(i)$ is substituted by the value d_i of the tuple $\mathbf{d} = \langle d_1, \dots, d_n \rangle \in \|r_1\|$; in the special cases, we can use nullary functions (constants) for the expression e (for example, for the NULL value).
(note that r is empty if e is an expression and r_1 empty as well).
Then, for a nonempty relation r_1 , the EXTEND r_1 ADD $a, name$ AS e (i.e.,

$r_1(a, name, e)$ can be represented by the following logical equivalence:
 $r(x_1, \dots, x_{n+1}) \Leftrightarrow (r_1(x_1, \dots, x_n) \wedge (x_{n+1} = e(\mathbf{x})))$
 where $e(\mathbf{x})$ is obtained by substituting each $name_i = nr_{r_1}(j)$ (of the j -th column of r_1) in the expression e by the variable x_j .

We are able to define a new relation with a single tuple $\langle \bar{d}_1, \dots, \bar{d}_k \rangle, k \geq 1$ with the given list of attributes $\langle a_1, \dots, a_k \rangle$, by the following finite length expression,
 $EXTEND (... (EXTEND r_\emptyset ADD a_1, name_1 AS \bar{d}_1) ...) ADD a_k, name_k AS \bar{d}_k$, or equivalently by $r_\emptyset(a_1, name_1, \bar{d}_1) \otimes \dots \otimes r_\emptyset(a_k, name_k, \bar{d}_k)$, where r_\emptyset is the empty type relation with $\|r_\emptyset\| = \{ \langle \rangle \}$, $ar(r_\emptyset) = 0$ introduced in Definition 2, and empty functions atr_{r_\emptyset} and nr_{r_\emptyset} . Such single tuple relations can be used for an insertion in a given relation (with the same list of attributes) in what follows.
Update operators. The three update operators, 'UPDATE', 'DELETE' and 'INSERT' of the Relational algebra, are derived operators from these previously defined operators in the following way:

1. Each algebraic formulae 'DELETE FROM r WHERE C ' is equivalent to the formula ' r MINUS (r WHERE C)'.
2. Each algebraic expression (a term) 'INSERT INTO $r[S]$ VALUES (list of values)', 'INSERT INTO $r[S]$ AS SELECT...', is equivalent to ' r UNION r_1 ' where the union compatible relation r_1 is a one-tuple relation (defined by list) in the first, or a relation defined by 'SELECT...' in the second case.
 In the case of a single tuple insertion (version with 'VALUES') into a given relation r , we can define a single tuple relation r_1 by using 'EXTEND..' operations.
3. Each algebraic expression 'UPDATE r SET [$nr_r(i_1) = e_{i_1}, \dots, nr_r(i_k) = e_{i_k}$] WHERE C ', for $n = ar(r)$, where $e_{i_m}, 1 \leq i_m \leq n$ for $1 \leq m \leq k$ are the expressions and C is a condition, is equal to the formula ' $(r$ WHERE $\neg C$) UNION r_1 ', where r_1 is a relation expressed by
 $(EXTEND(... (EXTEND (r$ WHERE C) ADD $att_r(1), name_{e_1}$ AS e_1) ...) ADD $att_r(n), name_n$ AS e_n)[S],
 such that for each $1 \leq m \leq n$, if $m \notin \{i_1, \dots, i_k\}$ then $e_m = nr_r(m)$, and $S = \langle name_1, \dots, name_n \rangle$.

Consequently, all update operators of the relational algebra can be obtained by addition of these 'EXTEND _ ADD $a, name$ AS e ' operations.

Let us define the Σ_R -algebras as follows ([8], Definition 31 in Section 5.1):

Definition 6. We denote the algebra of the set of operations, introduced previously in this section (points from 1 to 6 and $EXTEND_ADD\ a, name\ AS\ e$) with additional nullary operator (empty-relation constant) \perp , by Σ_{RE} . Its subalgebra without $_MINUS_$ operator is denoted by Σ_R^+ , and without \perp and unary operators $EXTEND_ADD\ a, name\ AS\ e$ is denoted by Σ_R (it is the "select-project-join-rename+union" (SPJRU) subalgebra). We define the set of terms $\mathcal{T}_P X$ with variables in X of this Σ_R -algebra (and analogously for the terms $\mathcal{T}_P^+ X$ of Σ_R^+ -algebra), inductively as follows:
 1. Each relational symbol (a variable) $r \in X \subseteq \mathbb{R}$ and a constant (i.e., a nullary operation) is a term in $\mathcal{T}_P X$;
 2. Given any term $t_R \in \mathcal{T}_P X$ and an unary operation $o_i \in \Sigma_R$, $o_i(t_R) \in \mathcal{T}_P X$;

3. Given any two terms $t_R, t'_R \in \mathcal{T}_P X$ and a binary operation $o_i \in \Sigma_R$, $o_i(t_R, t'_R) \in \mathcal{T}_P X$.

We define the evaluation of terms in $\mathcal{T}_P X$, for $X = \mathbb{R}$, by extending the assignment $\| \cdot \| : \mathbb{R} \rightarrow \underline{\mathcal{I}}$, which assigns a relation to each relational symbol (a variable) to all terms by the function $\| \cdot \|_{\#} : \mathcal{T}_P \mathbb{R} \rightarrow \underline{\mathcal{I}}$ (with $\|r\|_{\#} = \|r\|$), where $\underline{\mathcal{I}}$ is the universal database instance (set of all relations for a given universe \mathcal{D}). For a given term t_R with relational symbols $r_1, \dots, r_k \in \mathbb{R}$, $\|t_R\|_{\#}$ is the relational table obtained from this expression for the given set of relations $\|r_1\|, \dots, \|r_k\| \in \underline{\mathcal{I}}$, with the constraint that $\|t_R \text{ UNION } t'_R\|_{\#} = \|t_R\|_{\#} \cup \|t'_R\|_{\#}$ if the relations $\|t_R\|_{\#}$ and $\|t'_R\|_{\#}$ are union compatible; $\perp = \{<>\} = \|r_{\emptyset}\|$ (empty relation) otherwise. We say that two terms $t_R, t'_R \in \mathcal{T}_P X$ are equivalent (or equal), denoted by $t_R \approx t'_R$, if for all assignments $\|t_R\|_{\#} = \|t'_R\|_{\#}$.

We say that an extension $\|t_R\|_{\#}$, of a term $t_R \in \mathcal{T}_P X$, is *vector relation* of the *vector view* denoted by $\vec{t_R}$ if the type of $\|t_R\|_{\#}$ is equal to the type of the vector relation r_V . Let $R = \|\vec{t_R}\|_{\#}$ be the relational table with the four attributes (as r_V) *r-name*, *t-index*, *a-name* and *value*, then its used-defined view representation can be derived as follows:

Definition 7. VIEW MATERIALIZATION: Let $t_R \in \mathcal{T}_P X$ be a user-defined SPJU (Select-Project-Join-Union) view over a database schema $\mathcal{A} = (S_A, \Sigma_A)$ with the type (the tuple of the view columns) $\mathfrak{S} = \langle (r_{k_1}, \text{name}_{k_1}), \dots, (r_{k_m}, \text{name}_{k_m}) \rangle$, where the i -th column $(r_{k_i}, \text{name}_{k_i})$ is the column with name equal to name_{k_i} of the relation name $r_{k_i} \in S_A$, $1 \leq i \leq m$, and $\vec{t_R}$ be the rewritten query over r_V . Let $R = \|\vec{t_R}\|_{\#}$ be the resulting relational table with the four attributes (as r_V) *r-name*, *t-index*, *a-name* and *value*. We define the operation VIEW of the transformation of R into the user defined view representation by:

VIEW(\mathfrak{S}, R) = $\{ \langle d_1, \dots, d_m \rangle \mid \exists ID \in \pi_3(R), \forall 1 \leq i \leq m, \langle r_{k_i}, \text{name}_{k_i}, ID, d_i \rangle \in R; \text{otherwise set } d_i \text{ to NULL} \}$.

Notice that we have $\|r_k\| = \text{VIEW}(\mathfrak{S}, R) = \text{MATTER}(r_k, R)$ for each $r_k \in S_A$ with $R = \bigcup_{d \in \|r_k\|} \text{PARSE}(r_k, d)$, and $\mathfrak{S} = \langle (r_k, nr_{r_k}(1)), \dots, (r_k, nr_{r_k}(ar(r_k))) \rangle$, and hence the nonSQL operation MATTER is a special case of the operation VIEW.

For any original user-defined query (term) t_R over a user-defined database schema \mathcal{A} , by $\vec{t_R}$ we denote the equivalent (rewritten) query over the vector relation r_V . We have the following important result for the IRDBs:

Proposition 2 *There exists a complete algorithm for the term rewriting of any user-defined SQL term t_R over a schema $\vec{\mathcal{A}}$, of the full relational algebra Σ_{RE} in Definition 6, into an equivalent vector query $\vec{t_R}$ over the vector relation r_V . If t_R is a SPJU term (in Definition 7) of the type \mathfrak{S} then $\|t_R\|_{\#} = \text{VIEW}(\mathfrak{S}, \|\vec{t_R}\|_{\#})$.*

Proof: In this proof we will use the convention that two NULL values can not be compared as equal, because their meaning is that the value is missing, and two missing values not necessarily are equal. In fact in r_V we do not store the null values but consider them as unknown missing values. Thus, when there are null values in the columns of the user-defined tables being joined, the null values do not match each other.

Let us show that there is such a query (relation algebra term) rewriting for each basic relational operator previously described, recursively (in what follows, if $t_R = r$ then $\vec{t}_R = r_V$ WHERE $r\text{-name} = r$):

1. (Rename). $t'_R = r$ RENAME $name_1$ AS $name_2$ where the result is identical to input argument (relation) r except that the column i with name $nr_r(i) = name_1$ in all tuples is renamed to $nr_r(i) = name_2$. The rewritten vector query is $\vec{t}'_R = \text{UPDATE } r_V \text{ SET [a-name = name}_2\text{] WHERE (r-name = r) } \wedge \text{(a-name = name}_1\text{)}$;
2. (Projection). $t'_R = t_R[S]$, where $S = \langle (r_{j_1}, nr_{r_{j_1}}(i_1)), \dots, (r_{j_k}, nr_{r_{j_k}}(i_k)) \rangle \subseteq \mathfrak{S}$, with $k \geq 1$ and $1 \leq i_m \leq ar(r_{j_m})$ for $1 \leq m \leq k$, is a subset of the type \mathfrak{S} of the term t_R . We define the rewritten vector query $\vec{t}'_R = t_R[S]$
 $= \vec{t}_R$ WHERE $((nr_{\vec{t}_R}(1) = r_{j_1}) \wedge (nr_{\vec{t}_R}(2) = nr_{r_{j_1}}(i_1))) \vee \dots \vee ((nr_{\vec{t}_R}(1) = r_{j_k}) \wedge (nr_{\vec{t}_R}(2) = nr_{r_{j_k}}(i_k)))$;
3. (Join). $t'_R = t_{R,1} \bowtie_S t_{R,2}$, where $S = (((r_{l_1}, nr_{r_{l_1}}(i_1)), (r_{n_1}, nr_{r_{n_1}}(j_1))), \dots, ((r_{l_m}, nr_{r_{l_m}}(i_m)), (r_{n_m}, nr_{r_{n_m}}(j_m))))$ with $1 \leq i_k \leq |\mathfrak{S}_1|$ and $1 \leq j_k \leq |\mathfrak{S}_2|$ for $1 \leq k \leq m$, where \mathfrak{S}_1 and \mathfrak{S}_2 are the types of $t_{R,1}$ and $t_{R,2}$, respectively. .
 Let us define the following relational algebra terms:

$r = \overrightarrow{t_{R,1}} \otimes \dots \otimes \overrightarrow{t_{R,1}} \otimes \overrightarrow{t_{R,2}} \otimes \dots \otimes \overrightarrow{t_{R,2}}$; (the first m are for the attributes of $\overrightarrow{t_{R,1}}$ in S , and the last m are for the corresponded joined attributes of $\overrightarrow{t_{R,2}}$ in S). Note that each column name of $\overrightarrow{t_{R,1}}$ will be renamed m times in order to have for each column different name in standard way as for attributes: for example, the column a-name in $\overrightarrow{t_{R,1}}$ will be repeated by a-name(1), ..., a-name(m) (and similarly for each column name of $\overrightarrow{t_{R,2}}$), and the attribute $at_r(2)$ (of the column t-index in r_V) will be repeated by its copies $at_r(2)(1), \dots, at_r(2)(2m)$ (in what follows will be also generated the $(2m+1)$ -th copy $at_r(2)(2m+1)$ in the algebra term t_2). Thus,

$t_1 = r$ WHERE $(\bigwedge_{1 \leq k \leq m} ((nr_r(4k-3) = r_{l_k}) \wedge (nr_r(4k-1) = nr_{r_{l_k}}(i_k)) \wedge (nr_r(4(m+k)-3) = r_{n_k}) \wedge (nr_r(4(m+k)-1) = nr_{r_{n_k}}(j_k)) \wedge (nr_r(4k) = nr_r(4(m+k)+4)))$
 $\wedge ((m=1) \vee ((nr_r(2) = nr_r(6) = \dots = nr_r(4m-2))$
 $\wedge (nr_r(4m+2) = nr_r(4m+6) = \dots = nr_r(8m-2))))$;
 $t_2 = (\text{EXTEND } t_1 \text{ ADD } ((atr_r(2))(2m+1), name_3, Hash(atr_r(1), atr_r(2), \dots, atr_r(8m))))[nr_{t_1}(2), nr_{t_1}(4m+2), name_3]$;

where $name_3$ is a fresh new name and

$\|t_2\|_{\#} = \{ \langle ID_1, ID_2, ID_3 \rangle \mid ID_1 \text{ is the tuple-index in } \|t_{R,1}\|_{\#} \text{ and } ID_2 \text{ is the corresponding joined tuple-index in } \|t_{R,2}\|_{\#}, \text{ while } ID_3 \text{ is the fresh new generated (by Hash function) tuple-index for the tuple obtained by join operation} \}$,

then for the Cartesian products $t_3 = \overrightarrow{t_{R,1}} \otimes t_2$ and $t_4 = \overrightarrow{t_{R,2}} \otimes t_2$,

$\vec{t}'_R = \overrightarrow{t_{R,1} \bowtie_S t_{R,2}}$

$$= ((t_3 \text{ WHERE } (nr_{t_3}(2) = nr_{t_3}(5))) [nr_{t_3}(1), name_3, nr_{t_3}(3), nr_{t_3}(4)]) \\ \text{UNION } ((t_4 \text{ WHERE } (nr_{t_4}(2) = nr_{t_4}(6))) [nr_{t_4}(1), name_3, nr_{t_4}(3), nr_{t_4}(4)]);$$

4. (Selection). $t'_R = t_R \text{ WHERE } C$:

4.1 When a condition C is a finite-length logical formula that consists of atoms $'(r_{i_1}, name_{i_1}) \theta (r_{j_1}, name_{j_1})'$ or $'(r_{i_1}, name_{i_1}) \theta \bar{d}'$ or $'(r_{i_1}, name_{i_1}) \text{ NOT NULL}'$ with built-in predicates $\theta \in \Sigma_\theta \supseteq \{=, >, <\}$, a constant \bar{d} , and the logical operators, between the columns in the type \mathfrak{S} of the term t_R .

The condition C , composed by $k \geq 1$ different columns in \mathfrak{S} , we denote by $C((r_{i_1}, name_{i_1}), \dots, (r_{i_k}, name_{i_k}))$, $k \geq 1$, and hence we define the rewritten vector query

$$\overrightarrow{t'_R} = \overrightarrow{t_R \text{ WHERE } C} = \overrightarrow{t_R \text{ WHERE } nr_{\overrightarrow{t_R}}(2) \text{ IN } t_1}$$

where for $r = \overbrace{\overrightarrow{t_R} \otimes \dots \otimes \overrightarrow{t_R}}^k$ we define the unary relation which contains the tuple-indexes of the relation $\|t_R\|$ for its tuples which satisfy the selection condition C , by the following selection term

$$t_1 = (r \text{ WHERE } ((nr_r(1) = r_{i_1} \wedge nr_r(3) = name_{i_1}) \wedge \dots \wedge (nr_r(1 + 4(k-1)) = r_{i_k} \wedge nr_r(3 + 4(k-1)) = name_{i_k})) \wedge C(nr_r(4), \dots, nr_r(4k)) \\ \wedge ((k=1) \vee (nr_r(2) = nr_r(6) = \dots = nr_r(2 + 4(k-1)))) [nr_r(2)];$$

4.2 Case when $C = (r_{i_1}, name_{i_1}) \text{ NULL}$,

$$\overrightarrow{t'_R} = \overrightarrow{t_R \text{ WHERE } (r_{i_1}, name_{i_1}) \text{ NULL}} = \overrightarrow{t_R \text{ WHERE } nr_{\overrightarrow{t_R}}(2) \text{ NOT IN } t_2},$$

$$\text{where } t_2 = (\overrightarrow{t_R} \text{ WHERE } ((nr_{\overrightarrow{t_R}}(1) = r_{i_1}) \wedge (nr_{\overrightarrow{t_R}}(3) = name_{i_1}))) [nr_{\overrightarrow{t_R}}(2)].$$

From the fact that $t_R \text{ WHERE } C_1 \wedge C_2 = (t_R \text{ WHERE } C_1) \text{ WHERE } C_2$ and $t_R \text{ WHERE } C_1 \vee C_2 = (t_R \text{ WHERE } C_1) \text{ UNION } (t_R \text{ WHERE } C_2)$, and De Morgan laws, $\neg(C_1 \wedge C_2) = \neg C_1 \vee \neg C_2$, $\neg(C_1 \vee C_2) = \neg C_1 \wedge \neg C_2$, we can always divide any selection in the components of the two disjoint cases above;

5. (Union). $t'_R = t_{R,1} \text{ UNION } t_{R,2}$, where R is a table $\{\langle r_{l_k}, name_{l_k}, r_{n_k}, name_{n_k} \rangle \mid 1 \leq k \leq m\}$ such that $\mathfrak{S}_1 = \langle (r_{l_1}, name_{l_1}), \dots, (r_{l_m}, name_{l_m}) \rangle$ and $\mathfrak{S}_2 = \langle (r_{n_1}, name_{n_1}), \dots, (r_{n_m}, name_{n_m}) \rangle$ are the types of $t_{R,1}$ and $t_{R,2}$, respectively, with the union-compatible columns $\langle r_{l_k}, name_{l_k} \rangle$ and $\langle r_{n_k}, name_{n_k} \rangle$ for every $1 \leq k \leq m$.

We define the relational algebra term $t_1 = \overrightarrow{t_{R,2}} \otimes R$, so that

$$\overrightarrow{t'_R} = \overrightarrow{t_{R,1} \text{ UNION } t_{R,2}} \\ = \overrightarrow{t_{R,1} \text{ UNION } ((t_1 \text{ WHERE } ((nr_{t_1}(1) = nr_{t_1}(7)) \wedge (nr_{t_1}(3) = nr_{t_1}(8)))) [nr_{t_1}(5), nr_{t_1}(2), nr_{t_1}(6), nr_{t_1}(4)]},$$

so that the relation-column names of the union will be equal to the column names of the first term in this union;

6. (Set difference). $t'_R = t_{R,1} \text{ MINUS } t_{R,2}$, where R is a table $\{\langle r_{l_k}, nr_{r_{l_k}}(i_k), r_{n_k}, nr_{r_{n_k}}(j_k) \rangle \mid 1 \leq k \leq m\}$ such that $\mathfrak{S}_1 = \langle (r_{l_1}, nr_{r_{l_1}}(i_1)), \dots, (r_{l_m}, nr_{r_{l_m}}(i_m)) \rangle$ and $\mathfrak{S}_2 = \langle (r_{n_1}, nr_{r_{n_1}}(j_1)), \dots, (r_{n_m}, nr_{r_{n_m}}(j_m)) \rangle$ are the types of $t_{R,1}$ and $t_{R,2}$, respectively, with the union-compatible columns $\langle r_{l_k}, nr_{r_{l_k}}(i_k) \rangle$ and $\langle r_{n_k}, nr_{r_{n_k}}(j_k) \rangle$ for every $1 \leq k \leq m$. Let us define the following relational algebra terms:

$$\begin{aligned}
r &= \overrightarrow{t_{R,1}} \otimes \dots \otimes \overrightarrow{t_{R,1}} \otimes \overrightarrow{t_{R,2}} \otimes \dots \otimes \overrightarrow{t_{R,2}}; \text{ (the first } m \text{ are for the attributes} \\
&\text{of } \overrightarrow{t_{R,1}} \text{ in } \mathfrak{S}_1, \text{ and the last } m \text{ are for the corresponded joined attributes of } \overrightarrow{t_{R,2}} \text{ in} \\
&\mathfrak{S}_2). \text{ Thus,} \\
t_1 &= (r \text{ WHERE } (\bigwedge_{1 \leq k \leq m} ((nr_r(4k-3) = r_{l_k}) \wedge (nr_r(4k-1) = nr_{r_{l_k}}(i_k)) \wedge \\
&(nr_r(4(m+k)-3) = r_{n_k}) \wedge (nr_r(4(m+k)-1) = nr_{r_{n_k}}(j_k)) \wedge (nr_r(4k) = \\
&nr_r(4(m+k)+4))) \\
&\wedge ((m=1) \vee ((nr_r(2) = nr_r(6) = \dots = nr_r(4m-2)) \\
&\wedge (nr_r(4m+2) = nr_r(4m+6) = \dots = nr_r(8m-2)))))) [nr_{t_1}(2)]; \\
&\text{where } \|t_1\|_{\#} = \{\langle ID_1 \rangle \mid ID_1 \text{ is the tuple-index of a tuple in } \|t_{R,1}\|_{\#} \text{ for which} \\
&\text{there exists an equal tuple in } \|t_{R,2}\|_{\#}\}. \text{ Then,} \\
\overrightarrow{t'_R} &= \overrightarrow{t_{R,1}} \text{ MINUS } \overrightarrow{t_{R,2}} = \overrightarrow{t_{R,1}} \text{ WHERE } nr_{\overrightarrow{t_{R,1}}}(2) \text{ NOT IN } t_1.
\end{aligned}$$

It is easy to show that for the cases from 2 to 6, we obtain that $\|t'_R\|_{\#} = \underline{VIEW}(\mathfrak{S}, \|t'_R\|_{\#})$, where \mathfrak{S} is the type of the relational algebra term t'_R . Thus, for any SPJU term t_R obtained by the composition of these basic relational algebra operators we have that $\|t_R\|_{\#} = \underline{VIEW}(\mathfrak{S}, \|t_R\|_{\#})$.

The update operators are rewritten as follows:

1. (Insert). INSERT INTO $r[S]$ VALUES (d_1, \dots, d_m) , where $S = \langle nr_r(i_1), \dots, nr_r(i_m) \rangle$, $1 \leq m \leq ar(r)$, is the subset of mutually different attribute names of r and all v_i , $1 \leq i \leq m$ are the values different from NULL. It is rewritten into the following set of terms:
 $\{\text{INSERT INTO } r_V[\text{x-name, t-index, a-name, value}] \text{ VALUES } (r, Hash(d_1, \dots, d_m), nr_r(i_k), d_k) \mid 1 \leq k \leq m\}.$
 Note that before the execution of this set of insertion in r_V , the DBMS has to control if it satisfy all user-defined integrity constraints in the user-defined database schema \mathcal{A} ;
2. (Delete). DELETE FROM r WHERE C , is rewritten into the term:
 DELETE FROM r_V WHERE t-index IN $\overrightarrow{t_R}[nr_{\overrightarrow{t_R}}(2)]$,
 where $\overrightarrow{t_R} = r$ WHERE \overrightarrow{C} is the selection term as described in point 4 above;
3. (Update). the existence of the rewriting of this operation is obvious, from the fact that it can always be decomposed as deletion and after that the insertion of the tuples.

□

This proposition demonstrates that the IRDB is full SQL database, so that each user-defined query over the used-defined RDB database schema \mathcal{A} can be equivalently transformed by query-rewriting into a query over the vector relation r_V . However, in the IRDBMSs we can use more powerful and efficient algorithms in order to execute each original user-defined query over the vector relation r_V .

Notice that this proposition demonstrates that the IRDB is a kind of GAV Data Integration System $\mathcal{I} = (\mathcal{A}, \mathcal{S}, \mathcal{M})$ in Definition 1 where we do not materialize the user-defined schema \mathcal{A} but only the vector relation $r_V \in \mathcal{S}$ and each original query $q(\mathbf{x})$

over the empty schema \mathcal{A} will be rewritten into a vector query $\overrightarrow{q(\mathbf{x})}$ of the type \mathfrak{S} over the vector relation r_V , and then the resulting view $\text{VIEW}(\mathfrak{S}, \|\overrightarrow{q(\mathbf{x})}\|_{\#})$ will be returned to user's application.

Thus, an IRDB is a member of the NewSQL, that is, a member of a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (read-write) workloads while still maintaining the ACID guarantees of a traditional database system.

6 Conclusion

The method of parsing of a relational instance-database A with the user-defined schema \mathcal{A} into a vector relation \overrightarrow{A} , used in order to represent the information in a standard and simple key/value form, today in various applications of Big Data, introduces the intensional concepts for the user-defined relations of the schema \mathcal{A} . In Tarskian semantics of the FOL used to define the semantics of the standard RDBs, one defines what it takes for a sentence in a language to be true relative to a model. This puts one in a position to define what it takes for a sentence in a language to be valid. Tarskian semantics often proves quite useful in logic. Despite this, Tarskian semantics neglects meaning, as if truth in language were autonomous. Because of that the Tarskian theory of truth becomes inessential to the semantics for more expressive logics, or more 'natural' languages.

Both, Montague's and Bealer's approaches were useful for this investigation of the intensional FOL with intensional abstraction operator, but the first is not adequate and explains why we adopted two-step intensional semantics (intensional interpretation with the set of extensionalization functions). Based on this intensional extension of the FOL, we defined a new family of IRDBs. We have shown that also with this extended intensional semantics we may continue to use the same SQL used for the RDBs.

This new family of IRDBs extends the traditional RDBs with new features. However, it is compatible in the way how to present the data by user-defined database schemas (as in RDBs) and with SQL for management of such a relational data. The structure of RDB is parsed into a vector key/value relation so that we obtain a column representation of data used in Big Data applications, covering the key/value and column-based Big Data applications as well, into a unifying RDB framework.

Note that the method of parsing is well suited for the migration from all existent RDB applications where the data is stored in the relational tables, so that this solution gives the possibility to pass easily from the actual RDBs into the new machine engines for the IRDB. We preserve all metadata (RDB schema definitions) without modification and only dematerialize its relational tables by transferring their stored data into the vector relation r_V (possibly in a number of disjoint partitions over a number of nodes). From the fact that we are using the query rewriting IDBMS, the current user's (legacy) applications does not need any modification and they continue to "see" the same user-defined RDB schema as before. Consequently, this IRDB solution is adequate for a massive migration from the already obsolete and slow RDBMSs into a new family of fast, NewSQL schema-flexible (with also 'Open schemas') and Big Data scalable IRDBMSs.

References

1. COMPUTERWORLD, "No to SQL? Anti-database movement gains steam," http://www.computerworld.com/s/article/9135086/No_to_SQL_Anti_database_movement_gains_steam, June, 2009.
2. M.Stonebraker and C.Ugur, "One size fits all," In *ICDE 05 Proceedings, Washington, DC, USA: IEEE Computer Society*, pp. 2–11, 2005.
3. M.Stonebraker, S.Madden, D.Abadı, S.Harizopoulos, N.Hachem, and P.Helland, "The end of and architectural era: (it's time for a complete rewrite)," In *VLDB 07 Proceedings, VLDB Endowment*, pp. 1150–1160, 2007.
4. R.Kallman, H.Kimura, J.Natkins, A.Pavio, S.Zdonik, D.J.Abadı, E.P.C.Jones, S.Madden, A.Rasin, M.Stonebraker, Y.Zhang, and J.Hugg, "H-store: a high-performance, distributed main memory transaction processing system," In *Proceedings of the VLDB 08, VLDB Endowment*, pp. 1496–1499, 2008.
5. M.Stonebraker, "SQL databases v. NoSQL databases," *Communications of the ACM* 53:4, 2010.
6. M.Stonebraker, D.J.Abadı, D.J.Dewitt, S.Madden, E.Paulson, A.Pavio, and A.Rasin, "MapReduce and parallel DBMSs: Friends or Foes?," *Communications of the ACM* 53:64, *Doi:10.1145/1629175.1629197*, 2010.
7. A.Pavio, E.Paulson, D.J.Abadı, D.J.Dewitt, S.Madden, A.Rasin, and M.Stonebraker, "A comparison of approaches to large-scale data analysis," In *Proceedings of the 35th SIGMOD Conference, ACM Press, New York*, pp. 165–178, 2009.
8. Z. Majkić, "Big Data Integration Theory," *Springer-Verlag, Texts in Computer Science, New York Inc.*, pp.516, 2014.
9. A.K.Chandra and D.Harel, "Structure and complexity of relational queries," *J.Comput.Syst.Science* 25,1, pp. 99–128, 1982.
10. M.Lenzerini, "Data integration:a theoretical perspective," 2002, pp. 233–246.
11. J.Pustejovsky and B.Boguraev, "Lexical knowledge representation and natural language processing," *Artificial Intelligence*, 63, pp. 193–223, 1993.
12. M.C.Fitting, "First-Order Intensional Logic," *Annals of Pure and Applied Logic* 127, pp. 171–193, 2004.
13. Z.Majkić, "First-order logic: Modality and intensionality," *arXiv: 1103.0680v1*, 03 March, pp. 1–33, 2011.
14. Z.Majkić, "Conservative intensional extension of Tarski's semantics," *Advances in Artificial Intelligence, Hindawi Publishing Corporation, ISSN: 1687-7470*, 23 October, pp. 1–17, 2012.
15. G.Bealer, "Universals," *The Journal of Philosophy*, vol. 90, pp. 5–32, 1993.
16. G.Bealer, "Theories of properties, relations, and propositions," *The Journal of Philosophy*, vol. 76, pp. 634–648, 1979.
17. E.F.Codd, "Relational completeness of data base sublanguages," in *Data Base Systems: Courant Computer Science Symposia Series 6, Englewood Cliffs, N.J.: Prentice Hall*, 1972.
18. D.K.Lewis, "On the plurality of worlds," *Oxford: Blackwell*, 1986.
19. R.Stalnaker, "Inquiry," *Cambridge,MA:MIT Press*, 1984.
20. R.Montague, "Universal grammar," *Theoria*, vol. 36, pp. 373–398, 1970.
21. R.Montague, "The proper treatment of quantification in ordinary English," *Approaches to Natural Language, in J.Hintikka et al.(editors), Reidel, Dordrecht*, pp. 221–242, 1973.
22. R.Montague, "Formal philosophy. selected papers of Richard Montague," in *R.Thomason (editor), Yale University Press, New Haven, London*, pp. 108–221, 1974.
23. Z.Majkić, "Intensional first-order logic for P2P database systems," *Journal of Data Semantics (JoDS XII), LNCS 5480, Springer-Verlag Berlin Heidelberg*, pp. 131–152, 2009.

24. Z.Majkić, "Intensional semantics for RDF data structures," *12th International Database Engineering & Applications Systems (IDEAS08), Coimbra, Portugal, 10-13 September, 2008*.
25. G.Frege, "Über Sinn und Bedeutung," *Zeitschrift für Philosophie und Philosophische Kritik*, pp. 22–50, 1892.
26. B.Russell, "On Denoting," *Mind*, XIV, Reprinted in *Russell, Logic and Knowledge*, pp. 479–493, 1905.
27. A.N.Whitehead and B.Russell, "Principia Mathematica," Vol. I, Cambridge, 1910.
28. G.Bealer, "Quality and concept," *Oxford University Press, USA*, 1982.
29. R.Carnap, "Meaning and Necessity," *Chicago*, 1947.